Syntax and Semantics of the Compositional Interchange Format for Hybrid Systems

D.E. Nadales Agut[a], D.A. van Beek[a], J.E. Rooda[a]

[a]*Department of Mechanical Engineering, Eindhoven University of Technology (TU/e)*

Abstract

Different modeling formalisms for timed and hybrid systems exist, each of which addresses a specific set of problems, and has its own set of features. These formalisms and tools can be used in each stage of the embedded systems development, to verify and validate various requirements.

The Compositional Interchange Format (CIF), is a formalism based on hybrid automata, which are composed using process algebraic operators. CIF aims to establish interoperability among a wide range of formalisms and tools by means of model transformations and co-simulation, which avoids the need for implementing many bilateral translators.

This work presents the syntax and formal semantics of CIF. The semantics is shown to be compositional, and proven to preserve certain algebraic properties, which express our intuition about the behavior of the language operators. In addition we show how CIF operators can be combined to implement widely used constructs present in other timed and hybrid formalisms, and we illustrate the applicability of the formalism by developing several examples.

Based on the formal specification of CIF, an Eclipse based simulation environment has been developed. We expect this work to serve as the basis for the formal definition of semantic preserving transformations between various languages for the specification of timed and hybrid systems.

Keywords: hybrid systems, formal semantics, structured operational semantics

Email addresses: `D.E.Nadales.Agut@tue.nl` (D.E. Nadales Agut), `D.A.v.Beek@tue.nl` (D.A. van Beek), `J.E.Rooda@tue.nl` (J.E. Rooda)

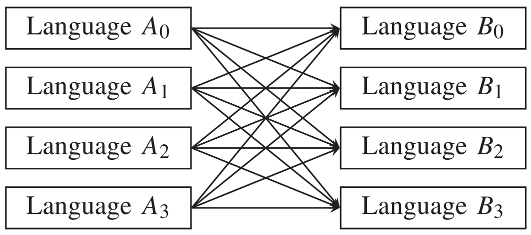


Figure 1: Multiple model transformations without an interchange format.

1. Introduction

1.1. Background

Controller software has become an artifact that is present in a wide range of embedded systems. Embedded software programs must interact with physical components, and contain a high degree of parallelism. In addition, they must meet critical safety, liveness, and performance requirements. All this makes the design of embedded software a difficult task, which calls for a model based approach that enables validation and verification via modeling.

Embedded systems combine the discrete essence of the software, and continuous aspects of the physical environment. Therefore, timed and hybrid formalisms [**? ? ?**], which are designed to combine computational, timed, and physical aspects of a system in one formal model, arise as the natural candidates for specification, verification, and validation of embedded system software.

But there is no panacea: different modeling formalisms for timed and hybrid systems exist. Each of these formalisms addresses a specific set of problems, and has its own set of features. Moreover, several formalisms and tools can be used in each stage of the embedded systems development, to verify and validate various requirements. This led to the need for integrated tool support for the design of large complex controlled systems, from the first concept to the implementation, and further on, over their entire life cycle.

The Compositional Interchange Format (CIF), is a formalism based on hybrid automata, which are composed using process algebraic operators. CIF aims to establish interoperability among a wide range of formalisms (and tools) by means of model transformations to and from CIF. In this way, the implementation of many bi-lateral translators between specific formalisms can be avoided, as shown in Figs. 1 and 2.

CIF, being an interchange format, has a number of distinctive features [**? ?**]:

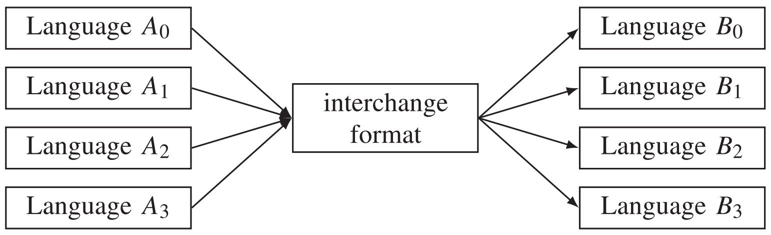


Figure 2: Multiple model transformations using an interchange format.

- It has a formal and compositional semantics, and thus it allows definition and proofs of property-preserving model transformations.
- It has concepts based on mathematics, which are independent of implementation aspects such as numerical equation sorting algorithms.
- It supports arbitrary differential algebraic equations.
- It incorporates concepts from hybrid automata theory and process algebra, such as parallel composition, different kinds of urgency and synchronization by means of shared variables and shared actions.
- It supports modularity, allowing to declare actions and variables local to a module.
- It supports large scale systems modeling by means of parameterized process definition and instantiation (reuse, hierarchy). Recently, CIF was extended with superstates [**?**]. Such an extension is not considered in this work.

All concepts in CIF have a formal and compositional semantics. As stated in Section **??**, a formal semantics has several advantages: it gives a precise meaning to CIF specifications, it facilitates the precise specification of models, it provides a reference against which implementations can be judged, and it enables the formal definition and proof of semantics-preserving model transformations. In addition, theoretical results are only possible if they rely on a formal semantics.

The semantics of CIF is defined via structured operational semantics (SOS) [**?**] rules. The reason for using operational semantics in an automaton-based framework is that model transformations to and from CIF are not only executed on "complete" models, but also on components of bigger models. Thus, it is crucial that the CIF semantics is compositional, which we ensure by requiring bisimulation (equivalence) to be a congruence for all the CIF constructs. This is guaranteed using the process-tyft format of [**?**].

1.2. Related work

Related work on interchange formats for hybrid systems is the following:

- A Hybrid System Interchange Format (HSIF) [**?**], in the MoBIES project.
- An abstract semantics of an interchange format based on the Metropolis meta model [**?**], as a continuation of the COLUMBUS project [**?**].
- An interchange format for switched linear systems [**?**] in the form of piecewise affine system (PWAs), in the HYCON NoE [**?**] project.

An overview of these formalisms is given in [**?**]. Next we present a summary of this overview.

In HSIF, a network of hybrid automata is used for model representation. The network behaves as a parallel composition of its automata, without hierarchy or modules. Variables can be shared or local, and the communication mechanism is based on broadcasting of boolean signals, where signals are partitioned in input and output signals. Each signal is required to be either a global input to the network or to be modified by exactly one automaton. The semantics is defined only for "acyclic dependency graphs" with respect to the use of signals. The interchange automaton format defined in this article aims to be more general than HSIF, and does not incorporate tool limitations, such as restrictions on circular dependencies, or restrictions on shared variables and algebraic loops, in its compositional formal semantics.

The abstract semantics presented in [**?**], takes implementation considerations into account, such as equation sorting, iterations that may be required for state-event detection, and iterations for reaching a fixed-point in case of algebraic loops. The semantics of CIF, defined in Section **??**, defines mathematically the semantics of hybrid automata, in a compositional way, independently of tool limitations and implementation aspects, such as equation sorting or event detection.

In the HYCON NoE [**?**] an interchange format for switched linear systems was defined [**?**]. This interchange format is based on piecewise affine (PWA) systems. Several tools, based on among others PWA, HYSDEL, MLD (see [**?**] for an overview relating these languages) have been connected to this interchange format. CIF is a much more general interchange format. The relation between PWA systems and (linear) hybrid automata is defined in [**?**].

1.3. Overview of work on CIF

Figure 3 gives a concise overview of past and present work on CIF in several European and national (Dutch) projects, in particular the HYCON and HYCON2 networks of excellence [**? ?**], the ITEA2 Twins project [**?**], the national Darwin

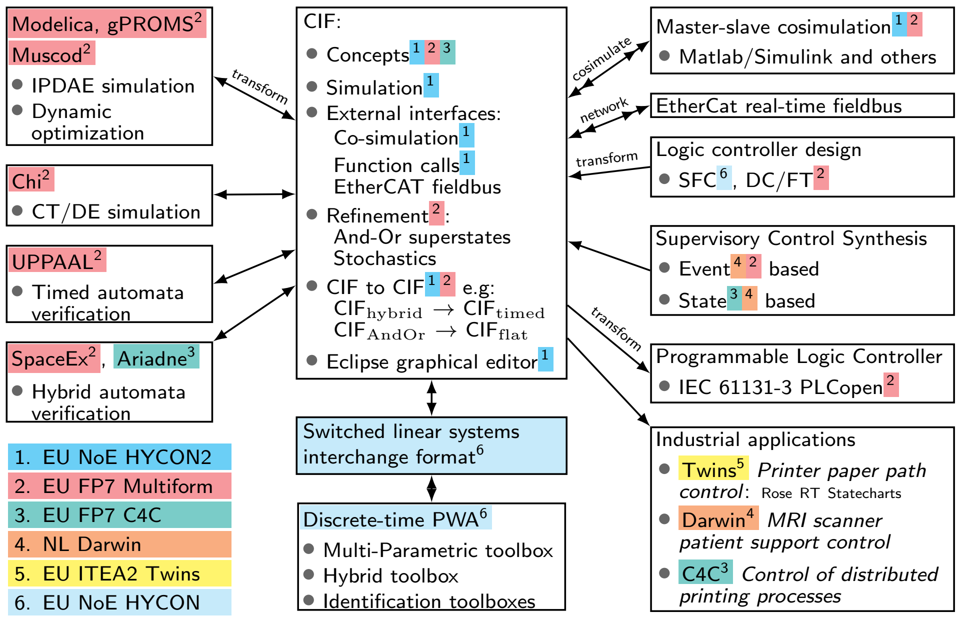


Figure 3: Overview of work on CIF.

project [?], and the FP7 C4C and Multiform projects [? ?]. The following languages and tools from Figure 3 are currently connected to CIF:

1. Modelica and gPROMS: For IPDAE (Integral and Partial Differential Algebraic Equations) modeling and simulation, translations between CIF and the advanced modeling languages Modelica and gPROMS are available [? ?]. This significantly increases the applicability of CIF in industrial practice.
2. MUSCOD-II: A connection of continuous CIF models to the dynamic simulation environment MUSCOD-II has been developed. This allows to interface to optimization-based synthesis tools [?].
3. Uppaal: A semantic preserving transformation from CIF to Uppaal has been defined in [?]. The implementation is described in [?]. The CIF-to-Uppaal transformation has been used, for example, to verify liveness properties of a synthesized supervisory controller of a patient support system of an MRI scanner as described under item 7.
4. SpaceEx: A transformation between CIF and SpaceEx was developed for model checking of hybrid CIF models [?].
5. Matlab/Simulink: A Matlab toolbox and a Simulink integration component were developed that allows to simulate CIF models directly from the Matlab command line, and in principle also from within Simulink models [?]. The

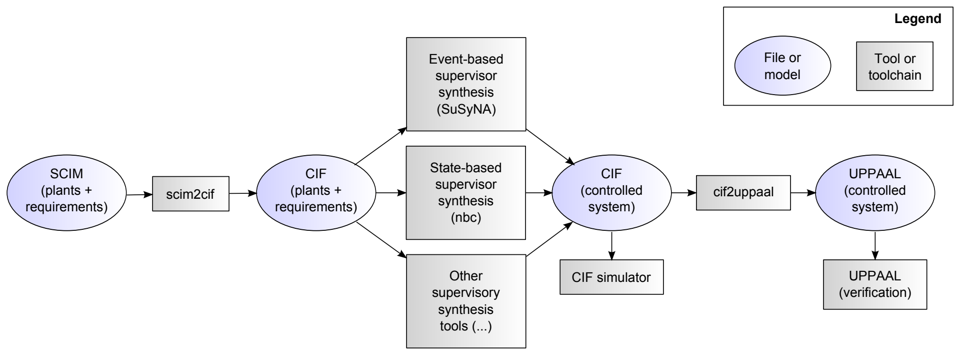

Legend
File or model
Tool or toolchain
SCIM (plants + requirements)
scim2cif
CIF (plants + requirements)
Event-based supervisor synthesis (SuSyNA)
State-based supervisor synthesis (nbc)
Other supervisory synthesis tools (...)
CIF (controlled system)
CIF simulator
cif2uppaal
UPPAAL (controlled system)
UPPAAL (verification)

Figure 4: Supervisory controller synthesis via CIF with verification in Uppaal.

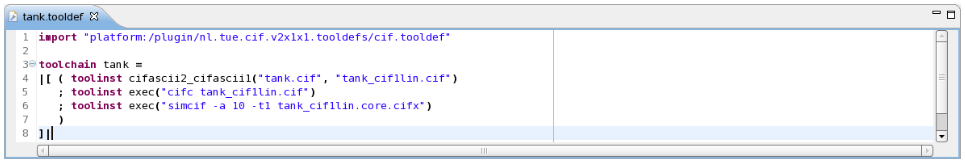
latter requires an additional S-function [**?**] interface.

6. Sequential Function Charts: A transformation of SFCs to CIF is defined in [**?**]. Sequential Function Charts is one of the languages defined in the IEC 61131-3 standard [**?**] for logic controller design.
7. Supervisory control synthesis tools: several transformations have been defined to allow different supervisory control synthesis tools to share the same CIF input specifications for plant models and (event-based) control requirement, and to share the same CIF output for the synthesized supervisory controllers. The generated CIF supervisory controllers can be used, among others, for simulation-based testing and for real-time control [**? ? ?**]. The CIF-based supervisory controller synthesis tool chain was tested by means of simulation and real-time control on an actual patient support system of an MRI scanner [**?**]. Figure 4 shows the automatic toolchain that has been used for controller synthesis. Supervisory control synthesis guarantees safety and nonblockingness by restricting behavior. By means of subsequent Uppaal verification, liveness properties can be checked. Note that the supervisory control requirements can also be defined by means of the visual Hierarchical Cause/Effect Charts editor, as described in [**?**].

Connection of the other tools from Figure 3 is still work in progress. Some of the connections of CIF to other tools shown in the figure, such as the connection to EtherCAT for real-time control [**?**], are still based on the old version of CIF, CIF1 [**?**]. New developments are based on CIF2 [**? ? ?**]

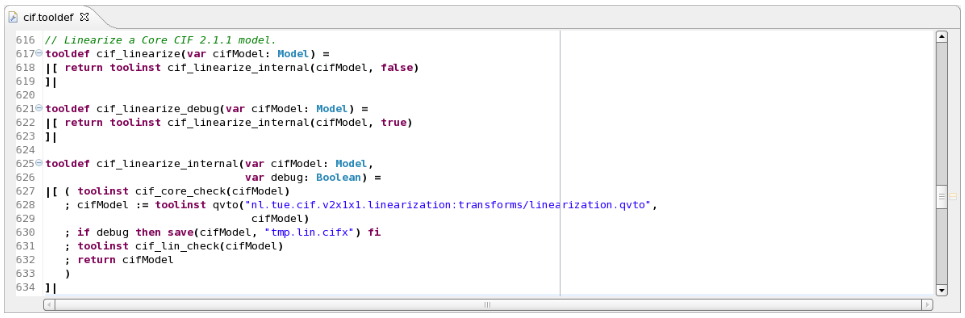
The transformation framework used for model transformations to and from CIF is based on OMG [**?**] standards: Class diagrams, in the form of Ecore metamodels, are used for the conceptual definition of CIF; QVTo is used as the model to model transformation language; and Acceleo is used as the model to text (code) transformation language [**?** **?**].

The use of a transformation language, such as QVTo, has as advantages that transformations are specified at the problem domain instead of coded at the implementation level, that implementation efforts are reduced and transformations are more robust for changes. For defining chains of transformations, the ToolDef [**?**] language has been defined. A syntax aware ToolDef editor has been generated by means of EMFText [**?**]. The editor provides among others integrated parsing, static semantic checking, syntax and occurrence highlighting, and code folding, see the examples in Figures 5 and 6.



```
tank.tooldef

import "platform:/plugin/nl.tue.cif.v2x1x1.tooldefs/cif.tooldef"

toolchain tank =
|[ ( toolinst cifascii2_cifascii1("tank.cif", "tank_cif1lin.cif")
   ; toolinst exec("cifc tank_cif1lin.cif")
   ; toolinst exec("simcif -a 10 -t1 tank_cif1lin.core.cifx")
   )
]|
```

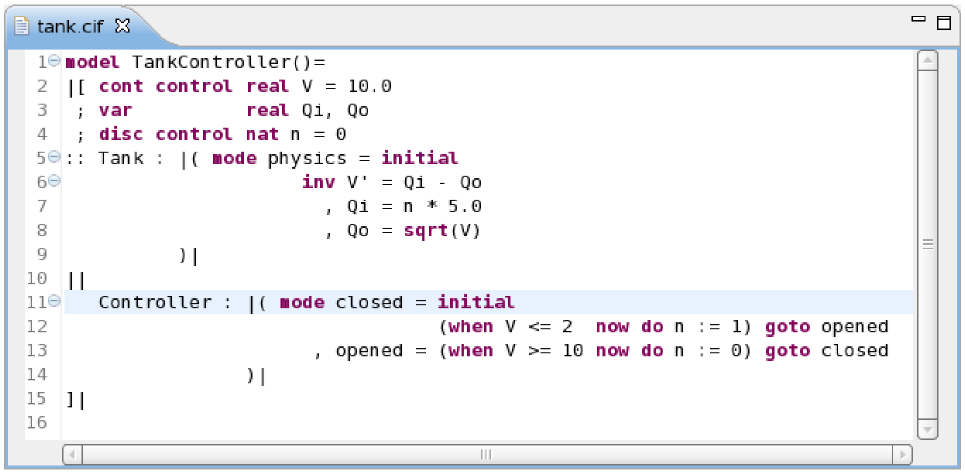
Figure 5: ToolDef tool chaining example.



```
cif.tooldef

// Linearize a Core CIF 2.1.1 model.
tooldef cif_linearize(var cifModel: Model) =
|[ return toolinst cif_linearize_internal(cifModel, false)
]|

tooldef cif_linearize_debug(var cifModel: Model) =
|[ return toolinst cif_linearize_internal(cifModel, true)
]|

tooldef cif_linearize_internal(var cifModel: Model,
                               var debug: Boolean) =
|[ ( toolinst cif_core_check(cifModel)
   ; cifModel := toolinst qvto("nl.tue.cif.v2x1x1.linearization:transforms/linearization.qvto",
                               cifModel)
   ; if debug then save(cifModel, "tmp.lin.cifx") fi
   ; toolinst cif_lin_check(cifModel)
   ; return cifModel
   )
]|
```

Figure 6: ToolDef tool specification example.

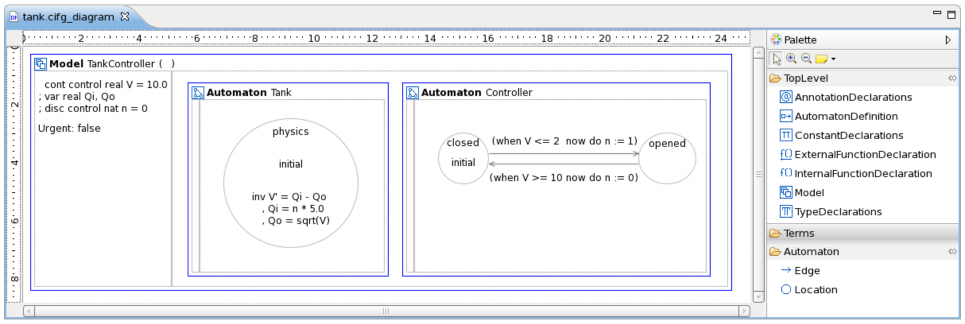
Beside being used for model transformations, CIF can also be used as a stand-alone modeling and simulation formalism for timed and hybrid systems. An Eclipse-based [**?**] tool set has been developed that provides a user-friendly environment for simulation, analysis, design, and for the definition of model transformations [**?**]. The CIF simulator was developed on the basis of the SOS (structured operational semantics) specification of the language [**?**]. To support hierarchical model development via stepwise refinement, CIF was recently extended with and-or superstates, leading to *hierarchical CIF* (HCIF) [**?**]. Simulation of HCIF models is possible via a HCIF to CIF transformation, which eliminates the and-or superstates by flattening the model. CIF models can be expressed using a textual notation (Figure 7), or a graphical notation (Figure 8). Simulation runs can be visu-



```
model TankController()=
|[ cont control real V = 10.0
 ; var           real Qi, Qo
 ; disc control nat n = 0
:: Tank : |( mode physics = initial
                     inv V' = Qi - Qo
                       , Qi = n * 5.0
                       , Qo = sqrt(V)
           )|
||
   Controller : |( mode closed = initial
                                 (when V <= 2  now do n := 1) goto opened
                      , opened = (when V >= 10 now do n := 0) goto closed
                 )|
]|
```

Figure 7: CIF textual editor

alized using a Scalable Vector Graphics visualizer (SVG) [?] as shown in Figure **??**. Most of the CIF tools are open source, and available via [?].



Model TankController ( )

```
 cont control real V = 10.0
; var real Qi, Qo
; disc control nat n = 0
Urgent: false
```

Automaton Tank

```
physics
initial
inv V' = Qi - Qo
  , Qi = n * 5.0
  , Qo = sqrt(V)
```

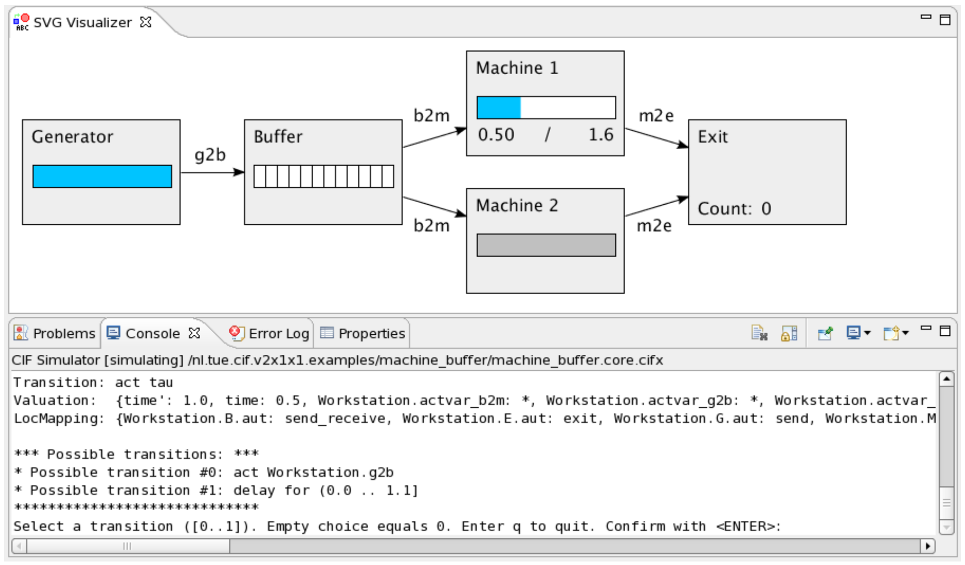
Automaton Controller

```
closed
initial
(when V <= 2  now do n := 1)
opened
(when V >= 10 now do n := 0)
```

Figure 8: CIF visual editor

1.4. *Objective*

The goal of this paper is to present the core concepts of CIF, and their specification by means of a formal semantics. In doing so, novel operators are introduced, such as synchronization and control. The semantics is validated by showing that it is compositional, and that it preserves certain algebraic properties, which express our intuition about the behavior of the language operators. In addition we show how CIF operators can be combined to implement widely used constructs present in other timed and hybrid formalisms.



CIF Simulator [simulating] /nl.tue.cif.v2x1x1.examples/machine_buffer/machine_buffer.core.cifx

```
Transition: act tau
Valuation:  {time': 1.0, time: 0.5, Workstation.actvar_b2m: *, Workstation.actvar_g2b: *, Workstation.actvar_
LocMapping: {Workstation.B.aut: send_receive, Workstation.E.aut: exit, Workstation.G.aut: send, Workstation.M

*** Possible transitions: ***
* Possible transition #0: act Workstation.g2b
* Possible transition #1: delay for (0.0 .. 1.1]
******************************
Select a transition ([0..1]). Empty choice equals 0. Enter q to quit. Confirm with <ENTER>:
```

Figure 9: CIF SVG visualizer.

The structure of the paper is as follows. Section **??** presents the CIF syntax, both informally and formally. To be able to explain CIF semantics, and to get a better insight of the concepts involved, Section **??** develops the semantic framework. In Section **??** the language is explained informally, and the formal semantics is given in Section **??**. The semantics is validated in Section **??**. In Section **??**, CIF is illustrated by means of several examples. Concluding remarks are presented in Section **??**.

2. Syntax of CIF

This section presents the syntax of CIF. The basic building blocks of CIF are *automata*. They resemble the hybrid automata as presented in [**?**], which model computational and physical behavior of a system by mixing automata theory with the theory of differential algebraic equations.

Informally, a basic CIF automaton is shown in Figure **??**, which models a conveyor belt carrying bottles to be filled. Graphically, the name of the automaton ($Conveyor$) is specified in a box, placed above the top-left corner of the rectangle that encloses the drawing of the automaton. The volume of the bottle (V_B) cannot exceed the maximum allowed volume (V_{max}). The rate at which liquid enters the bottle is represented by variable Q. A clock c is used to ensure that Δ time units will elapse between the filling of two bottles, where Δ is some positive constant.

The automaton consists of two *locations*, $Moving$ and $Filling$, which are depicted

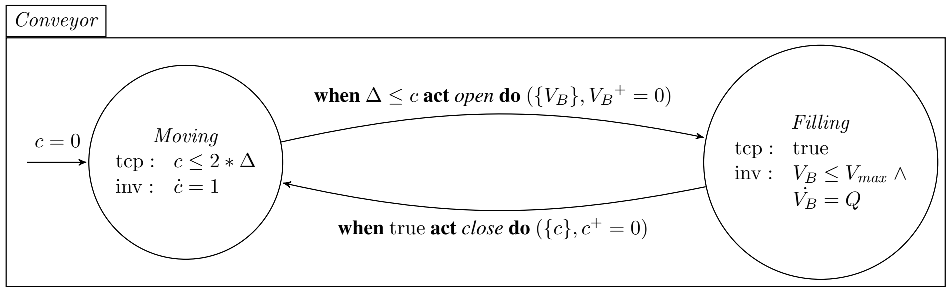

Conveyor
$c = 0$
$Moving$
tcp : $c \leq 2 * \Delta$
inv : $\dot{c} = 1$
when $\Delta \leq c$ **act** *open* **do** $(\{V_B\}, {V_B}^+ = 0)$
$Filling$
tcp : true
inv : $V_B \leq V_{max} \wedge$
$\dot{V_B} = Q$
when true **act** *close* **do** $(\{c\}, c^+ = 0)$

Figure 10: Model of a conveyor (without operators).

as circles. Locations represent the computational states of a system. Every location contains a predicate called *invariant*, which must hold as long as the system is in that state; and a *time can progress* (tcp) predicate, which must hold during time delays. In the example of Figure **??**, location $Moving$ has the predicate $\dot{c} = 1$ as invariant, and the predicate $c \leq 2 * \Delta$ as tcp. Location $Filling$ has the invariant $V_B \leq V_{max} \wedge \dot{V_B} = Q$, and the tcp predicate true. Invariants can be used, for instance, to specify differential algebraic equations. In this way, it is possible to model the physical behavior of a system in a particular computational state. Local urgency conditions can be defined using tcp predicates.

Edges represent discrete changes in the computational state of a system. An edge has a *source* and a *target* location, and its execution results in a change of location (unless the edge is a self loop). The automaton of Figure **??** has two edges, which are depicted as arrows (the arrows point to the target location). Every edge contains a predicate called *guard* ($\Delta \leq c$ and true, respectively, in Figure **??**) that determines under which conditions a transition can be executed, a predicate called *update* ($V_B^+ = 0$ and $c^+ = 0$, respectively, where x^+ denotes the value of variable x in the next state) that determines how the model variables change after performing the action, and a set of *jumping variables* ($\{V_B\}$ and $\{c\}$) that specify the variables that are changed by the action. Edges are labeled by *actions* ($open$ and $close$ in Figure **??**) that may be used to synchronize the behavior of automata in a parallel composition. Finally, every location has an *initialization predicate* associated to it, which describes constraints that the initial values of variables must satisfy if execution is to start in that location. Note that this predicate can be used to specify the initial locations of an automaton. In Figure **??**, location $Moving$ has $c = 0$ as its initial condition (depicted as an small incoming arrow without source location), and location $Filling$ has the predicate false as initial condition (depicted by the absence of such an incoming arrow), which means that execution cannot begin in that computational state. Section **??** explains the semantics of CIF automata in detail.

Following CIF concrete syntax, an arrow from location v to v' labeled **when** g **act** a **do** (W, v), represents the edge $(v, g, a, (W, r), v')$. If the guard declaration is omitted (**when** g part) then it is assumed to be true. If the action is omitted (**act** a part), it is assumed to be the silent action τ. If the update is omitted (**do** (W, r)) part, then it is assumed to be $(\emptyset, \mathrm{true})$.

Additional components of an automaton (not shown in the example presented here) include: a set of *control variables*, a set of *synchronizing actions*, and a *dynamic type* mapping. Intuitively, control variables are those variables that can only be modified by the automaton that declares them, and they do not change arbitrarily after performing an action. The set of synchronizing actions is used to specify which actions are to be synchronized when the automaton is composed in parallel. The concept of dynamic types is used to model constraints in the joint evolution of a variable and its dotted version. In CIF a dynamic type is a set containing pairs of functions, whose domain is a closed range of the form $[0, t]$, with $t \in \mathbb{T}$ (see Definition **??**). Notation $\mathbb{T}$ is used to refer to the set of all time points. Examples of dynamic types used in this work include *discrete*, *continuous*, and *clock*. Informally, if a variable x is of type discrete, then its value must remain constant in time delays, and $\dot{x}$ is always zero. On the other hand, the value of a continuous variable changes as a continuous function of time, and its dotted version represents its derivative. If a variable c is of type clock, then during time delays its value evolves (as a function of time) at rate 1. Section **??** provides more details. The syntax of these components is given in Definition **??**, and their associated concepts are explained in Section **??**.

Formally, the locations of CIF automata are taken from the set $\mathcal{L}$. Actions belong to the set $\mathcal{A}$. We use the symbol τ ($\tau \notin \mathcal{A}$) to refer to the silent action, and we define $\mathcal{A}_\tau \triangleq \mathcal{A} \cup \{\tau\}$. We distinguish the following types of variables: regular variables, denoted by the set $\mathcal{V}$; the dotted versions of those variables, which belong to the set $\dot{\mathcal{V}} \triangleq \{\dot{x} \mid x \in \mathcal{V}\}$; and *step variables*, which belong to the set $\{x^+ \mid x \in \mathcal{V} \cup \dot{\mathcal{V}}\}$. Variables are constrained by equations and we implement them as predicates. The values of the variables belong to the set Λ that contains the sets of booleans $\mathbb{B}$, and reals $\mathbb{R}$, among others. Guards are taken from the set $\mathcal{P}_g$; initialization predicates, invariants, and tcp predicates are taken from the sets $\mathcal{P}_t$; and update predicates are taken from the set $\mathcal{P}_r$. Expressions are taken from the set $\mathcal{E}$.

To formally characterize the set of all dynamic types, we need to define some operators on trajectories [**?**]. In the context of the current work, a trajectory is a function that maps time points onto valuations, and a valuation is a function that maps variables onto values. Given a trajectory ρ with domain $[0, t]$, and a time point s, $s \leq t$, the *prefix operator* returns a trajectory that equals ρ up to s. Similarly, given a time point s, the *postfix operator* allows to construct a valuation that equals ρ in the time interval $[s,t]$. The formal definition of these operators is given below, where given a function f, and a set A, $f\restriction_A$ is the restriction of f to A.

Definition 1 (Trajectory prefix operator)**.** *Given a time point $s \in \mathbb{T}$, and a trajectory $\rho : [0,t] \to A$, such that $s \leq t$, function $\rho^{\leq s}$ is defined by means of the following equality:*

$$\rho^{\leq s} \triangleq \rho \restriction_{[0,s]}$$

Definition 2 (Trajectory postfix operator)**.** *Given a time point $s \in \mathbb{T}$, and a trajectory $\rho : [0,t] \to A$, such that $0 \leq s \leq t$, function $\rho^{\geq s} : [0,t-s] \to A$ is defined by means of the following equality:*

$$\rho^{\geq s}(r) \triangleq \rho(r+s)$$

for all $r \in [0,t-s]$.

For the automaton postfix operator note that $\rho^{\geq s}(0) = \rho(s)$ and $\rho^{\geq s}(t-s) = \rho(t)$.

Finally, the *concatenation operator* allows to form a new trajectory by gluing two trajectories together.

Definition 3 (Concatenation operator)**.** *Given two trajectories $\rho : [0,t] \to A$ and $\rho' : [0,t'] \to A$, such that $\rho(t) = \rho'(0)$, function $\rho \cdot_t \rho' : [0,t+t'] \to A$ is defined as follows:*

$$(\rho \cdot_t \rho')(s) \triangleq \begin{cases} \rho(s) & \textit{if } s \leq t \\ \rho'(s-t) & \textit{if } t < s \end{cases}$$

Using these definition we can now formally define the set of all dynamic types, where we use $A \rightharpoonup B$ to denote the set of partial functions from A to B, and $\mathrm{dom}(f)$ is the domain of function f.

Definition 4 (Dynamic Types)**.** *The set $\mathcal{D} \subseteq 2^{(\mathbb{T} \rightharpoonup \Lambda) \times (\mathbb{T} \rightharpoonup \Lambda)}$ of dynamic types is the least set that satisfies for all $G \in \mathcal{D}$:*

1. *for all $(f,g) \in G$ such that $\mathrm{dom}(f) = [0,t]$, and for all $s \leq t$, $(f^{\leq s}, g^{\leq s}) \in G$.*
2. *for all $(f,g) \in G$ such that $\mathrm{dom}(f) = [0,t]$, and for all $0 \leq s \leq t$, $(f^{\geq s}, g^{\geq s}) \in G$.*
3. *for all $(f,g) \in G$ and $(f',g') \in G$ such that $\mathrm{dom}(f) = [0,t]$, $\mathrm{dom}(g) = [0,s]$, $f(t) = f'(0)$ and $g(s) = g'(0)$, we have $(f \cdot_t f', g \cdot_s g') \in G$.*

Such a definition of dynamic types is required to prove the properties of prefix and postfix closure, and the property of state. Analogous restrictions are required in the setting of hybrid I/O automata [**?**].

The exact syntax and semantics of predicates and expressions are left as a parameter of our theory, as we are not interested in the computational semantics of CIF in this paper. We do require however a minimum set of properties that have to be satisfied by the predicates. In the examples presented here, and in the tool implementations of CIF, $\mathcal{P}_g$, $\mathcal{P}_t$, and $\mathcal{P}_r$ are terms of the language of predicate logic [**?**], where for $\mathcal{P}_g$ and $\mathcal{P}_t$ the variables are taken from the set $\mathcal{V} \cup \dot{\mathcal{V}}$, and for $\mathcal{P}_r$, the variables are taken from the set $\mathcal{V} \cup \dot{\mathcal{V}} \cup \{x^+ \mid x \in \mathcal{V} \cup \dot{\mathcal{V}}\}$. As for expressions, the set $\mathcal{E}$ is usually instantiated with the set of arithmetic expressions. Given an expression e and a value v, we assume $e = v$ to be an element of $\mathcal{P}_i$, $i \in \{g, t, r\}$; and we assume that the predicates are closed under conjunction. Using these preliminaries, a CIF automaton is defined next.

Definition 5 (Automaton)**.** *An automaton is a tuple*

$$(V, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, E, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype})$$

where $V \subseteq \mathcal{L}$ *is the set of locations;* $\mathrm{init}, \mathrm{inv}, \mathrm{tcp}\colon V \to \mathcal{P}_t$ *are functions that associate to each location its corresponding initialization predicate, invariant, and tcp predicate, respectively;* $E \subseteq V \times \mathcal{P}_g \times \mathcal{A}_\tau \times (2^{\mathcal{V} \cup \dot{\mathcal{V}}} \times \mathcal{P}_r) \times V$ *is the set of edges;* $\mathrm{var}_C \subseteq \mathcal{V}$ *is the set of control variables;* $\mathrm{act}_S \subseteq \mathcal{A}$ *is the set of synchronizing actions; and* $\mathrm{dtype} \in \mathcal{V} \rightharpoonup \mathcal{D}$ *is the dynamic type mapping.*

Starting from an automaton, more complex models can be constructed by means of different operators. These include *parallel composition*, to model concurrent execution of systems; the *synchronization operator*, to declare synchronizing actions in a parallel composition; the *initialization operator*, to specify the initial conditions of a system; the *variable scope operator*, to declare identifiers as local; the *urgency operator* to declare actions as urgent; the *dynamic type operator* to associate dynamic types to variables; and the *control variable operator* to declare variables as controlled. Section **??** presents in detail the informal semantics of these operators. We use the term *composition* to refer to a model that contains zero or more of these operators. The set $\mathcal{C}$ refers to the set of all compositions, and is formally defined as follows.

Definition 6 (Compositions)**.** *The set of all compositions is defined by the abstract grammar of Table* **??**. *Where* $a \in \mathcal{A}$, $u \in \mathcal{P}_t$, $x \in \mathcal{V}$, $e \in \mathcal{E} \cup \{\bot\}$ *(*$\bot$ *is used to denote the undefined value),* $a_\tau \in \mathcal{A}_\tau$, *and* $G \in \mathcal{D}$.

The next sections explain, informally and formally, the semantic of CIF models.

$\mathcal{C} ::=$	α	automaton
$\mid$	$\mathcal{C} \parallel \mathcal{C}$	parallel composition
$\mid$	$\mathrm{ctrl}_x(\mathcal{C})$	control variable operator
$\mid$	$\upsilon_{a_\tau}(\mathcal{C})$	urgency operator
$\mid$	$\mathrm{D}_{x:G}(\mathcal{C})$	dynamic type operator
$\mid$	$u \gg \mathcal{C}$	initialization operator
$\mid$	$\gamma_a(\mathcal{C})$	synchronization operator
$\mid$	$\|[_{\mathrm{V}}\, x = e, \dot{x} = e :: \mathcal{C}\ ]\|$	variable scope operator
$\mid$	$\|[_{\mathrm{A}}\, a :: \mathcal{C}\ ]\|$	action scope operator

Table 1: CIF abstract grammar.

3. Semantic framework

In this section, the semantic framework is set up, which allow us to properly explain the semantics of CIF. First we present the concepts of variable valuations and flow trajectories. Next we describe the hybrid transition systems, which are used to model the semantics of CIF compositions. Finally, a formal definition of this semantic model is given.

3.1. Preliminaries

Semantically, the *execution* of a model causes changes to the values of the variables appearing in it. Thus, in the semantic framework it is necessary to represent the values of the variables in a particular instant. For this purpose, we use the concept of *valuation*, which is standard in semantics of processes with data. A valuation $\sigma \colon (\mathcal{V} \cup \dot{\mathcal{V}}) \to \Lambda$ is a function that for each variable returns its corresponding value. We use notation $\Sigma \triangleq (\mathcal{V} \cup \dot{\mathcal{V}}) \to \Lambda$ to refer to the set of all valuations.

Even though predicates are abstract entities, we assume that a satisfaction relation $\sigma \models u$ is defined, which expresses that predicate $u \in \mathcal{P}_t \cup \mathcal{P}_g \cup \mathcal{P}_r$ is satisfied (i.e. it evaluates to true) in valuation σ. For predicate logic, this relation can be defined in a standard way (see [**?**] for example). For a valuation σ, we define $\sigma^+ \triangleq \{(v^+, c) \mid (v, c) \in \sigma\}$.

In CIF, the values of variables change as the result of the execution of discrete actions, or as the result of time delays. The edges of the automata determine how the values of variables change after performing an action, and invariants specify how the values of variables evolve as time passes.

To model the evolution on the values of variables during time delays we use the concept of *variable trajectories*. A variable trajectory is a function $\rho \colon \mathbb{T} \rightharpoonup \Sigma$ that returns the valuations of the variables at each time point. In other words, $\rho(s)(x)$ is the value of variable x at time s along trajectory ρ. We assume the domain of variable trajectories to be closed intervals, i.e. intervals of the form $[0, t]$, where $t \in \mathbb{T}$ and $0 < t$.

In the examples presented here, and in the semantic rules for dynamic types, we refer to the evolution of a particular variable during time delays. From a variable trajectory ρ it is possible to reconstruct the evolution of a variable during the time delay that spans $\mathrm{dom}(\rho)$. Given a flow ρ, the evolution of a variable x during time delay $\mathrm{dom}(\rho)$ can be seen as a function $\rho_x : \mathrm{dom}(\rho) \to \Lambda$, such that for each time point s we have $\rho_x(s) \triangleq \rho(s)(x)$. In this way, $\rho_x(s)$ represents the value of variable x at time point s. Having presented the basic concepts of the semantic framework, we explain next the semantic model for CIF compositions.

3.2. Hybrid transition systems

The semantics of CIF compositions is given in terms of SOS rules, which induce hybrid transition systems (HTS) [**?**]. The states of the HTS are of the form (p, σ), where $p \in \mathcal{C}$ is a composition and σ is a valuation. As we stated earlier, valuations are used to capture the phenomenon of discrete change in the values of variables caused by the execution of actions in an automaton. There are three kind of transition in the HTS, namely, *action*, *time*, and *environment transitions*. We describe them in detail next.

Action transitions are of the form $(p, \sigma) \xrightarrow{a,b,X} (p', \sigma')$, and they model the execution of an action a by composition p in an initial valuation σ, which changes composition p into p' and results in a new valuation σ'. Label b is a boolean that indicates whether action a is synchronizing, and label X is the set of control variables[1] of p and p'. The term *active locations* is used to refer to the set of locations for which the initialization predicate holds. After an action is performed a unique active location is picked.

As an example, consider the conveyor belt automaton shown in Figure **??**, and the initial valuation $\{(c, \Delta + 1), (V_B, 7)\}$ (we use set notation for representing valuations, and only the relevant variables are shown). It can execute action $open$ since the guard is true. After executing the action the active location changes to $Filling$, which is represented by changing the init function so that it returns true for location $Filling$ and false for location $Moving$. The resulting automaton is shown in Figure **??**, and a possible new valuation is $\{(c, 15), (V_B, 0)\}$. Note that the value of c is changed arbitrarily after performing the transition. This is because variables in CIF are not controlled by default, which means that their value can change arbitrarily when a new value is not specified in the update predicate. If we denote the automaton of Figure **??** as $Conveyor[Moving]$, and the automaton of

[1]Control variables are explained in Section **??**.

Figure **??** as $Conveyor[Filling]$, this transition can be depicted as

$$(Conveyor[Moving], \{(c, \Delta + 1), (V_B, 7)\}) \xrightarrow{open, \text{false}, \emptyset} (Conveyor[Filling], \{(c, 15), (V_B, 0)\}).$$

Note that in the previous transition, the synchronization label is false because actions are non-synchronizing by default.

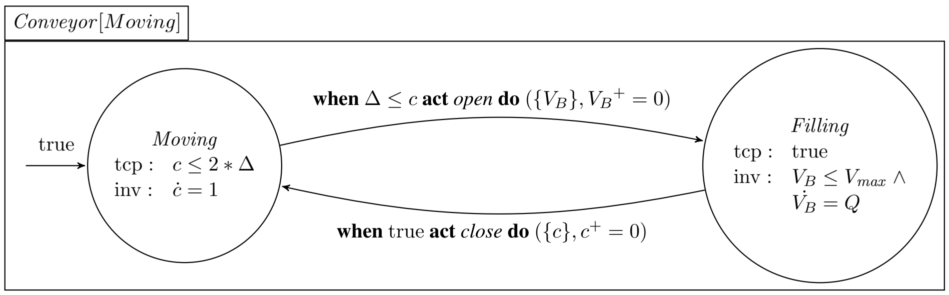

$Conveyor[Moving]$
true
Moving
tcp : $c \leq 2 * \Delta$
inv : $\dot{c} = 1$
when $\Delta \leq c$ **act** *open* **do** $(\{V_B\}, V_B^+ = 0)$
when true **act** *close* **do** $(\{c\}, c^+ = 0)$
Filling
tcp : true
inv : $V_B \leq V_{max} \wedge$ $\dot{V_B} = Q$

Figure 11: Model of a conveyor with initial predicate true.

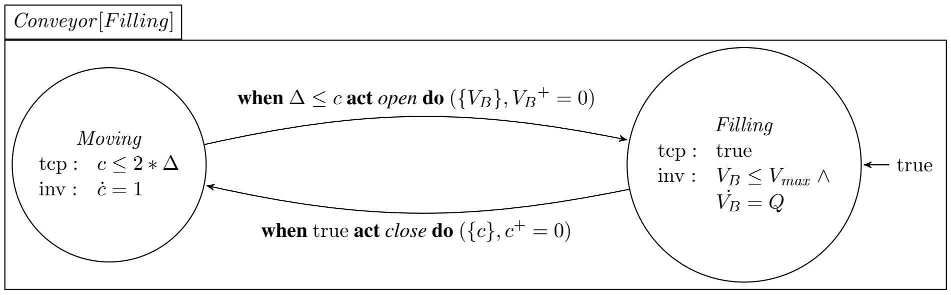

$Conveyor[Filling]$
Moving
tcp : $c \leq 2 * \Delta$
inv : $\dot{c} = 1$
when $\Delta \leq c$ **act** *open* **do** $(\{V_B\}, V_B^+ = 0)$
when true **act** *close* **do** $(\{c\}, c^+ = 0)$
Filling
tcp : true
inv : $V_B \leq V_{max} \wedge$ $\dot{V_B} = Q$
true

Figure 12: Conveyor in the filling state, after executing *open* action.

Time behavior is captured by *time transitions*. Time transitions are of the form $(p, \sigma) \overset{\rho, A, \theta}{\longmapsto} (p', \sigma')$, and they model the passage of time in composition p, in an initial valuation σ, which results in a composition p' and valuation σ'. Function ρ is the variable trajectory that models the evolution of variables during the time delay. Function $\theta : \mathbb{T} \rightharpoonup 2^{\mathcal{A}}$ is called *guard trajectory* [**?**], and it models the evolution of enabled actions during time delays. For each time point $s \in \text{dom}(\theta)$, the function application $\theta(s)$ yields the set of enabled actions of composition p at time s. For every time transition ρ, $\text{dom}(\rho) = [0, t]$, for some positive time point $t \in \mathbb{T}$, and $\text{dom}(\rho) = \text{dom}(\theta)$. Finally, label A contains the set of synchronizing actions of p and p' [2]. When time passes a *unique* location is picked.

Consider the automaton *Conveyor* (Figure **??**), in initial valuation $\{(c, 0), (V_B, 1)\}$. Suppose the automaton performs a time delay of 2 time units. Figure **??** shows a possible evolution of variables c and V_B during the time delay, i.e. ρ_c and ρ_{V_B}. Figure **??** shows the evolution of the guard trajectories, encoded in function θ. At the beginning there are no actions enabled, but after Δ ($\Delta \le 2$) time units, the set of enabled actions changes to $\{open\}$.

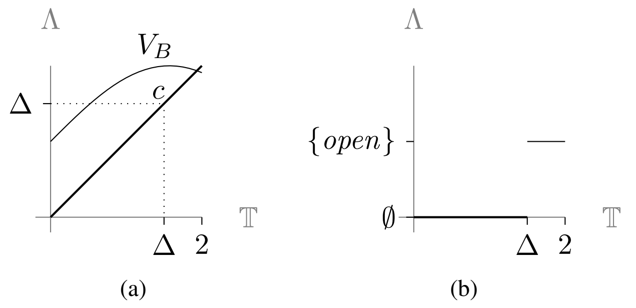


Figure 13: Example of trajectories during a time delay. **??** Variable trajectory. **??** Guard trajectory.

After the time delay, the init predicate of the automaton *Conveyor* is changed from $c = 0$ to true, which represents the fact that location *Moving* was picked. It is necessary to replace this predicate by true, otherwise, after the time delay shown here, the initialization predicate is false and the automaton has no initial locations. At the end of the delay, the new valuation becomes $\{(c, 2), (V_B, 1.9)\}$. The time transition that describes this delay is written as follows:

$$(Conveyor, \{(c, 0), (V_B, 1)\}) \overset{\rho,\emptyset,\theta}{\longmapsto} (Conveyor[Moving], \{(c, 2), (V_B, 1.9)\}).$$

To model parallel execution of compositions, we need the notion of *environment transitions*, which are transitions of the form $(p, \sigma) \overset{A}{\dashrightarrow} (p', \sigma')$, and they model the fact that composition p (p') is consistent (see Definition **??**) in valuation σ (σ'). Label A is the set of synchronizing actions of p and p'. Intuitively, environment transitions express which state changes by the environment are allowed. *Consistency* is defined recursively as follows.

Definition 7 (Consistency)**.** *Given a valuation σ, we define consistency as follows:*

[2]The set of synchronizing actions is not changed by transitions.

- *An automaton* $(V, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, E, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype})$ *is consistent in* σ *if there is a location* $v \in V$ *such that* $\sigma \models \mathrm{init}(v)$ *and* $\sigma \models \mathrm{inv}(v)$.

- *Composition* $p \parallel q$ *is consistent in valuation* σ *if* p *and* q *are consistent in valuation* σ.

- *Composition* $[\![_{\mathrm{V}}\, x = e_0, \dot{x} = e_1 :: p\,]\!]$ *is consistent in valuation* σ *if there are values* v_0 *and* v_1*, such that* $(\sigma \models (e_i = v_i)) \vee e_i = \bot$*,* $i \in \{0, 1\}$*, and* p *is consistent in valuation* $\{x \mapsto v_0, \dot{x} \mapsto v_1\} \succ \sigma$*, where, given two functions* $f : A \rightharpoonup C$ *and* $g : A \rightharpoonup C$*, function* $f \succ g : A \cup B \rightharpoonup C$ *is defined as follows:*
$$f \succ g(x) = \begin{cases} f(x) & \text{if } x \in \mathrm{dom}(f) \\ g(x) & \text{if } x \in \mathrm{dom}(g) \setminus \mathrm{dom}(f) \end{cases}$$
The variable scope operator $[\![_{\mathrm{V}}\, \_, \_ :: \_\,]\!]$ *is defined in Section* **??**.

- *For the remaining operators the definition of consistency is extended point-wise.*

We use notation $\sigma \models p$ to denote that composition p is consistent in valuation σ. Alternatively, we say that σ is consistent with p.

As an example, consider the automaton $\mathit{Conveyor}[\mathit{Filling}]$, depicted in Figure **??**, and initial valuation $\{(V_B, 4)\}$ (as in the previous examples we only show in the valuation the variables relevant for the example). Assume $V_{max} = 10$. Transition

$$(\mathit{Conveyor}[\mathit{Filling}], \{(V_B, 4)\}) \overset{\emptyset}{\dashrightarrow} (\mathit{Conveyor}[\mathit{Filling}], \{V_B, 8\})$$

models the fact that the value of V_B can be incremented by 4 units, given the fact that the invariant is preserved.

Having informally explained the basic ideas behind the HTS induced by the semantic rules, we give next the formal definition of these transition systems.

Definition 8 (Hybrid Transition System)**.** *A hybrid transition system (HTS) is a tuple of the form* $(Q, \mathcal{A}, \rightarrow, \longmapsto, \dashrightarrow)$ *where* $Q \triangleq \mathcal{C} \times \Sigma$*,* $\rightarrow\ \subseteq Q \times (\mathcal{A}_\tau \times \mathbb{B} \times 2^{\mathcal{A}}) \times Q$*,* $\longmapsto \subseteq Q \times ((\mathbb{T} \rightharpoonup \Sigma) \times 2^{\mathcal{A}} \times (\mathbb{T} \rightharpoonup 2^{\mathcal{A}})) \times Q$*, and* $\dashrightarrow \subseteq Q \times 2^{\mathcal{A}} \times Q$*.*

Even though fully formal, these definitions help to understand better the concepts to come. In the next sections, the semantics of CIF is explained informally, and in Section **??** formal definitions are provided.

4. Informal semantics

In this section we explain CIF by means of a bottle filling line, which is depicted in Figure **??**. It consists of a storage tank that is continuously filled with a flow Q_{in}, and a conveyor belt that supplies empty bottles.

We construct the example incrementally. First, the semantics of the conveyor belt model of Figure **??** is informally explained. Next, operators are added to modify its behavior in a convenient way. When we describe parallel composition, the model of the liquid tank is presented, which will complete the bottle filling line model.

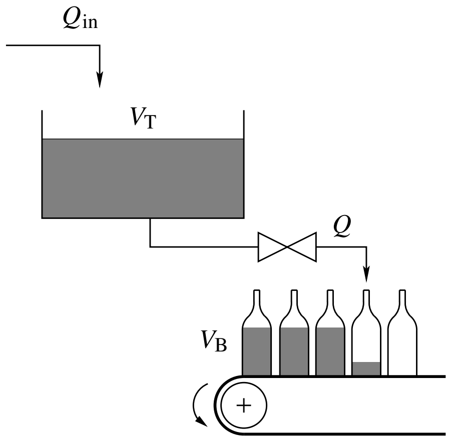


Figure 14: Bottle filling line.

4.1. Automata

In this section we explain informally the semantics of the automaton introduced in Figure **??** by showing a run of the system described by it.

Initially, the execution of the automaton can only begin in the location $Moving$, in an initial valuation in which $c = 0$. Assume $\Delta = 1.5$. This means that no action is enabled at the beginning, and only time can pass. Since the system is in location $Moving$, the value of variable $\dot{c}$ will evolve according to equation $\dot{c} = 1$, whereas the remaining variables can assume arbitrary values (remember that $\dot{c}$ is not defined to be the derivative of c yet).

Figure **??** shows one of the possible evolutions of the values of the variables c and $\dot{c}$ as a function of time (the other variables are not shown to keep the plot readable and the explanation simple). The time points at which discrete actions take place are marked in the x-axis with the name of the corresponding action, and a dotted vertical line. Notice that when actions are performed a variable can have two values: one before, and one after the action. This is because the values of variables change (jump) on action transitions. The x-axis represents the time points, and the y-axis the values. The curly braces above the plot show the locations where the automaton is at a given time.

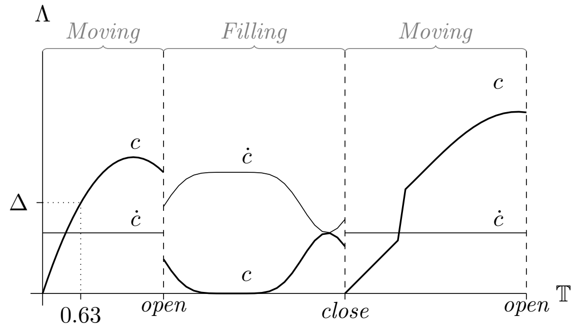


Figure 15: Conveyor run.

Figure **??** shows that during the time delay the value of $\dot{c}$ remains constant according to the equation $\dot{c} = 1$, and the value c can change arbitrarily (in the figure we have chosen a random function to represent this behavior). After a delay of 0.63 time units the guard $\Delta \leq c$ becomes enabled. However, in this particular run, the action is not executed immediately since actions are non-urgent by default.

At time 2 condition $\Delta \leq c$ still holds, and action *open* is executed. This causes a jump in the values of both c and $\dot{c}$. Since the update condition does not constrain the new values of these variables, they can take any values. However, the value of V_B (not shown in the figure) cannot change arbitrarily, since the update predicate enforces that $V_B = 0$ holds after the action.

Once in the location *Filling*, the system performs another delay. Here the values of c and $\dot{c}$ change randomly, since they are not constrained by any equation. However, the values of $\dot{V_B}$ and Q must be the same during this delay, since the evolution of these values has to satisfy the predicate (equation) $\dot{V_B} = Q$.

After a certain time, action *close* is executed. This causes the value of variable c to change to 0, as required by the update predicate. Note that the value of $\dot{c}$ has to change to 1 to satisfy the invariant. The values of the remaining variables can jump arbitrarily.

When location *Moving* is entered again the variables behave in a similar way as described before. Notice that we depict the evolution on the value of c during time as a continuous function, but since its behavior is not constrained, all kinds of trajectories are possible.

4.2. Dynamic type operator

In principle, during a time delay, the values of variables can change arbitrarily as long as they satisfy the invariants and tcp predicates, as we saw previously. This is not always desired. Consider the conveyor model, which has the variable $\dot{c}$, which represents the derivative of c. We do not want the values of c and $\dot{c}$ to evolve independently as time passes. That is, given a flow ρ, we would like function ρ_c to be a *derivable* function of time, and function $\rho_{\dot{c}}$ to be the *rate of change* of the value of ρ_c. Flows and invariants are not enough for ensuring this.

In hybrid formalisms, the evolution of the values of variables is constrained by declaring a variable to be of a certain *dynamic type* [**?**]. Informally a dynamic type specifies how the values of variables are allowed to change as time passes. The most common dynamic types that can be associated to a variable are *discrete*, *continuous*, and *algebraic*. The value of a discrete variable x remains constant in time delays, and $\dot{x}$ is the zero function. Given a continuous variable x, $\rho_{\dot{x}}$ is the derivative of ρ_x for any variable trajectory ρ. The values of algebraic variables can change arbitrarily during time delays.

In CIF, a dynamic type is a set of pairs of functions from time points to values. For instance the discrete dynamic type G_{disc} can be defined as follows:

$$\begin{aligned} G_{disc} \triangleq \{(f, g) \mid \langle \exists t : 0 < t : & \mathrm{dom}(f) = [0, t] = \mathrm{dom}(g) \wedge \\ & \langle \forall d, d' : d, d' \in [0, t] : f(d) = f(d') \wedge g(d) = 0 \rangle\rangle\} \end{aligned}$$

Similarly, the clock dynamic type G_{clock} is defined in the following way:

$$\begin{aligned} G_{clock} \triangleq \{(f, g) \mid \langle \exists t : 0 < t : & \mathrm{dom}(f) = [0, t] = \mathrm{dom}(g) \wedge \\ & \langle \forall d : d \in [0, t] : f(d) = f(0) + d \wedge g(d) = 1 \rangle\rangle\} \end{aligned}$$

And the continuous dynamic type G_{cont} can be defined as follows:

$$G_{\mathrm{cont}} \triangleq \{(f, g) \mid f \text{ is continuous in } [0, t] \wedge g \text{ is the derivative of } f \text{ in } [0, t]\}$$

Notice that, unlike other hybrid formalisms [**?**], the dynamic types of CIF do not constrain the discrete (jumping) behavior of a variable. Instead, we use the concept of control variables, as explained in Section **??**.

In the conveyor model example, we want to associate continuous dynamic types to variables c and V_B, since we would like predicates $\dot{c} = 1$ and $\dot{V_B} = Q$ to be interpreted as differential equations. On the other hand, we would like variable V_{max}, which represent the maximum volume of a bottle, to be a constant. Note that we do not pose any constraint on variable Q, as we will need this when putting the conveyor model in parallel with a tank model in Section **??**. According to the syntax of CIF, these dynamic types can be declared as shown in Equation **??**

$$\mathrm{D}_{\{c:G_{cont}, V_B:G_{cont}, V_{max}:G_{disc}\}}(Conveyor) \qquad (1)$$

where we use notation:

$$\mathrm{D}_{\{x_0:G_0,\ldots,x_{n-1}:G_{n-1}\}}(p)$$

as a way to abbreviate the nesting of several dynamic type operators

$$\mathrm{D}_{x_0:G_0}((\ldots\mathrm{D}_{x_{n-1}:G_{n-1}}(p)\ldots)).$$

Alternatively, these dynamic types can be specified as a part of the automaton. We have chosen to use operators instead, to explain the concepts in a stepwise manner. The usefulness of operators will be illustrated in Section **??**.

For the conveyor example, if we declare the dynamic type of variable c to be continuous then the evolutions shown in Figure **??** are no longer possible, since $\dot{c}$ does not represent the derivative of c. Figure **??** shows a possible evolution of the values of these variables after introducing the dynamic type.

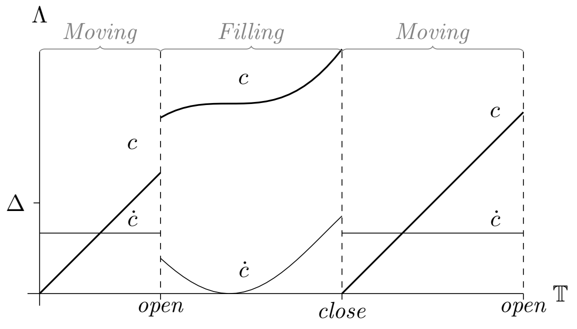


Figure 16: Conveyor run after introducing dynamic types.

4.3. Control variables

In CIF, the values of variables change arbitrarily after an action is executed, unless this is restricted by the update predicate associated to the action or by the invariants. It is often desirable that the model variables keep their values after an action is performed, as it is the case in programming languages, or timed and hybrid formalisms such as Uppaal [**?**] or Chi [**?**], respectively.

By declaring a variable as controlled, whenever an action is performed, the value of that variable will remain constant, unless it belongs to the set of jumping variables of the action. Furthermore, control variables can only be changed by the composition that declares them as controlled. In this way, the concept of control variables corresponds with the same concept in hybrid I/O automata [**? ?**].

To illustrate the effect of the control variable operator, we show in Equation **??** the model of the conveyor, henceforth denoted as $p_{Conveyor}$

$$\mathrm{ctrl}_{\{V_{max},V_B,c\}}(\mathrm{D}_{\{c:G_{cont},V_B:G_{cont},V_{max}:G_{disc}\}}(Conveyor)) \qquad (2)$$

where variables V_{max}, V_B, and c are declared as controlled. As with the dynamic type operator, we abbreviate the nested application of control variable operators using set notation. The behavior of variables c and $\dot{c}$ in this model is shown in Figure **??**. After action ($open$) is performed, the value of c cannot jump. Note that after performing action $close$, the value jumps to 0 because it is *explicitly* required by the update predicate of the edge that describes this action.

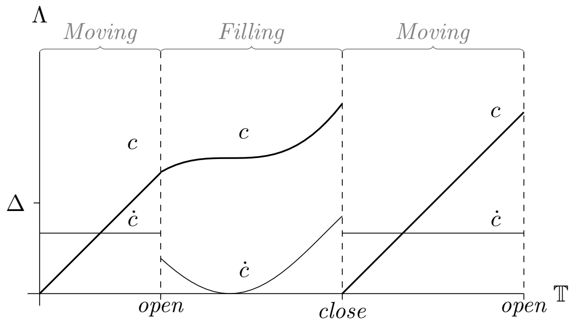


Figure 17: Conveyor run after introducing control variable operator.

4.4. Parallel composition

To model the parallel execution of systems, CIF provides the parallel composition operator. When two compositions p and q are put in parallel, they can execute synchronizing actions simultaneously and interleave asynchronous actions. They interact by means of shared variables. The passage of time synchronizes in both automata.

To illustrate parallel composition in CIF, we introduce first the model of a tank (without declarations), which is shown in Figure **??**. It has a unique initial location $Closed$, representing the state in which the valve of the tank is closed (and therefore the outgoing flow Q equals 0). In the $Closed$ state, the volume of the tank increases at rate Q_{in}, and its volume cannot exceed V_{Tmax}. When the tank is in the *Opened* state, the outgoing flow is set to Q_{set}, and the liquid enters the tank at rate $Q_{in} - Q$. In this state, the volume of the tank (V_T) cannot exceed the maximum value allowed either. If V_T reaches 0, then the $Empty$ state is reached, where the outgoing flow equals the incoming flow, and the system can linger in that state as long as the liquid volume is 0.

In the previous model, we declare all variables that are associated to differential equations, as continuous. These variables are V_T and Q. We want Q_{in} and Q_{set} to be constant (both after time delays, and after executing actions), thus we declare them as discrete and controlled. In addition, it is not desirable that the volume

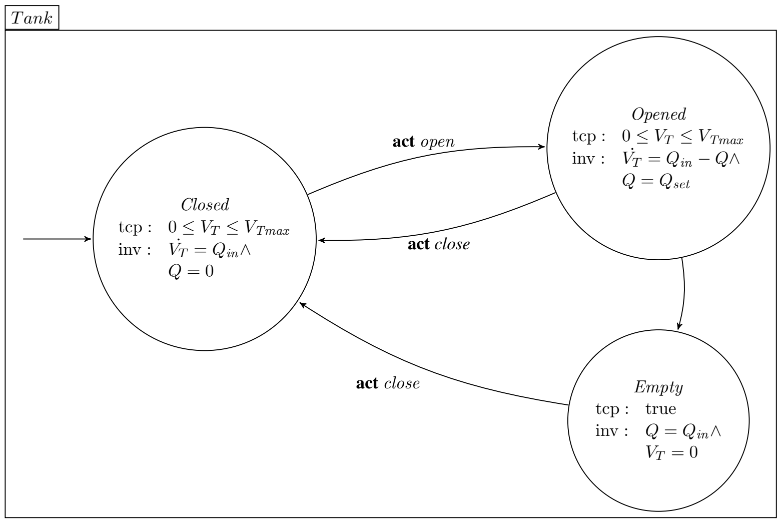

$Tank$
$Closed$
tcp : $0 \leq V_T \leq V_{Tmax}$
inv : $\dot{V_T} = Q_{in} \wedge$
$Q = 0$
$Opened$
tcp : $0 \leq V_T \leq V_{Tmax}$
inv : $\dot{V_T} = Q_{in} - Q \wedge$
$Q = Q_{set}$
$Empty$
tcp : true
inv : $Q = Q_{in} \wedge$
$V_T = 0$
act *open*
act *close*
act *close*

Figure 18: Tank model without declarations.

of the tank changes arbitrarily when we perform the $close$ action, and we do not want another component to modify the volume of the tank. Thus, we also declare V_T as controlled, and using a similar reasoning Q is made controlled as well. If we denote the automaton depicted in Figure **??** as $Tank$, then the model of the tank, henceforth referred to as p_{Tank}, with all the corresponding declarations can be written as shown in Equation **??**.

$$\text{ctrl}_{\{V_{Tmax}, Q_{in}, Q_{set}, V_T\}}(\text{D}_{\{V_T:G_{cont}, Q:G_{cont}, Q_{in}:G_{disc}, Q_{set}:G_{disc}\}}(Tank)) \tag{3}$$

Figure **??** shows the evolution of the values of variables in a simulation run. Initially Q can only be 0, whereas V_T can assume any value that is between 0 and V_{Tmax}. After a certain time delay, in which Q remains constant and V_T evolves according to $\dot{V_T} = Q_{in}$, action *open* is executed. Observe that the value of V_T is not changed after this action is performed (it is a control variable), whereas Q has to jump in order to satisfy equation $Q = Q_{set}$. In the second time delay, where the system is in the *Opened* state, the volume of the tank starts decreasing (we assume $Q_{in} < Q$) and Q has the same value as Q_{set} during the entire delay. When the volume of liquid in the tank reaches 0, further delays are not possible, since that would violate the tcp predicate, thus action τ has to be executed and the *Empty* state is entered. In this state, the tank remains empty ($V_T = 0$), and the outgoing flow decreases to the value of Q_{in} (we assume $Q_{in} < Q_{set}$). Finally, action $close$ is executed and the evolution of the variables in the last time delay shown in Figure **??** is as described before.

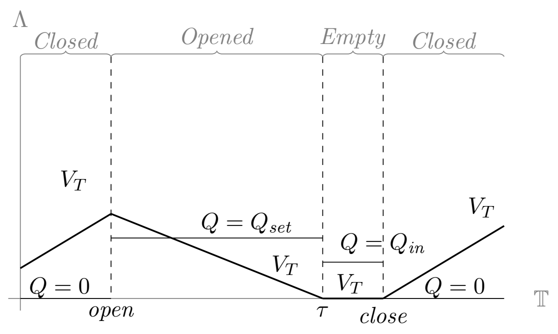


Figure 19: Run of the tank.

Next, we have to build the bottle filling line model. Thereto, we must put the models of Figures **??** and **??** ($p_{Conveyor}$ and p_{Tank} respectively) in parallel. However, this is not enough since actions are not synchronizing by default. Nevertheless, for illustration purposes, we show in Figure **??** a run of the system when these two models are put in parallel. We have depicted only variables relevant for our example. The uppermost braces show the states in which the conveyor model is at a given time, whereas the braces below these show the states of the tank. Initially the conveyor is moving and the tank is in the closed state. In the first time delay the values of variables V_T and Q evolve as described previously, whereas the values of variable V_B change arbitrarily in the first two time delays, and they are not shown in the plot. The first action that is executed is the *open* action in the tank, which does not synchronize with the action of the same name in the conveyor, which is executed later in time. When the conveyor performs its first action, variables V_T and Q are not affected, and the volume of liquid in the bottle (V_B) starts increasing at rate $Q = Q_{set}$. When the tank is emptied (V_T reaches 0), the tank performs a τ action, and in Figure **??** it is possible to see how this *does* affect the behavior of the conveyor model, since now the volume of liquid in the bottle increases but at a rate equal to Q_{in}. After a new delay, $V_B = V_{max}$ (remember V_{max} represents the maximum volume of the bottle). As a consequence no further time delays are allowed in the model, since in a parallel composition, time must be able to pass for all the components. Thus, action *close* executes in the conveyor. After a certain delay, the action *close* of the tank is performed, and the system starts behaving as described before.

The next section presents a model in which the actions *close* and *open* of the conveyor and the tank synchronize, yielding the desired synchronizing behavior for the bottle filling line model.

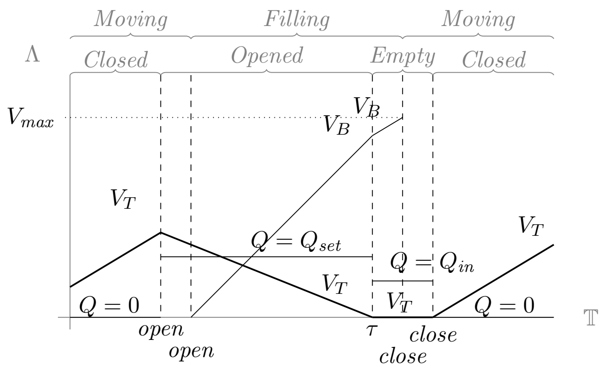


Figure 20: Run of conveyor and tank without explicit synchronization.

4.5. Synchronization operator

As we saw in the previous section, in CIF actions do not synchronize by default. This gives the modeller more flexibility as pointed out in [**?**], and it allows us to implement different communication patterns, as illustrated in Section **??**. Given a composition p, and an action a, if we want a to be executed synchronously when p is composed in parallel, we declare action a as synchronizing.

A composition $\gamma_a(p)$ declares action a to be synchronizing in p. When $\gamma_a(p)$ is put in parallel with a composition q, which also declares a as synchronizing, action a has to be executed by $\gamma_a(p)$ and q at the same time. Executions of a by only one of these components are not possible. When $\gamma_a(p)$ is put in parallel with a composition r, which does not declare a as synchronizing, only $\gamma_a(p)$ can execute a (independently of r).

As an example, consider the model of the conveyor depicted in Figure **??**, which we denote as $p_{Conveyor}$; and consider the model of the tank shown in Figure **??**, denoted as p_{Tank}. The behavior of composition $\gamma_{close}(\gamma_{open}(p_{Tank})) \parallel \gamma_{close}(\gamma_{open}(p_{Conveyor}))$ is illustrated in Figure **??**. Here it is possible to see that actions $open$ and $close$ must occur at the same time. Action $open$ cannot be executed earlier than Δ, because of the guard $\Delta \leq c$ in the edge of the conveyor. After this time, both models can execute this action synchronously. When the conveyor enters the state $Filling$, the volume of the liquid in the bottle starts rising, until it reaches V_{Tmax}, at which point time cannot progress any further, and action $close$ has to be executed in the tank and in the conveyor at the same time.

Note that composition $\gamma_{close}(\gamma_{open}(p_{Tank} \parallel p_{Conveyor}))$ does not give the desired behavior. If the models of tank and conveyor are composed in this way, actions $open$ and $close$ can be executed asynchronously. Section **??** provides the formal details.

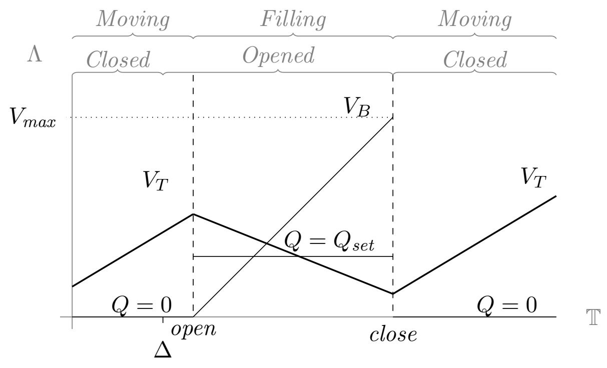


Figure 21: Run of conveyor and tank with explicit synchronization.

4.6. Urgency operator

In CIF models, actions are not urgent by default. This means that the execution of an action can be delayed as long as the equations in the model allow it. To change such behavior, urgency can be introduced by means of two concepts: time can progress predicates and invariants, and the urgency operator. As we have seen before, time can pass in an active location as long as its invariant and tcp predicates are satisfied. If these cannot be satisfied by any time delay, only actions can be executed, or the whole system deadlocks. In the conveyor model of Figure **??**, when the liquid in the bottle reaches its maximum allowed volume (V_{max}), time cannot progress, which makes action $close$ urgent.

Informally, an action is urgent if whenever it is enabled, time cannot progress[3]. For an automaton, urgency can be expressed by using tcp predicates. However, when parallel composition is involved, the definition of enabled actions becomes more complex, and urgency can no longer be defined through tcp predicates in a compositional way (see [**?**] and the example of Figure **??**). For this purpose, CIF has the urgency operator. Given a composition p and an action a, composition $\upsilon_a(p)$ expresses that no time delay is possible if action a is enabled in p.

The issue of defining urgency in a compositional way has been addressed in [**?**]. The authors define the semantics of timed systems in terms of timed transition systems. The definition of urgency for parallel composition is handled in syntactic way, by means of a composition operator. On the other hand, we define the semantics of urgency at the semantic level, using operations on the labels of the hybrid transitions (see Section **??**).

As an example, suppose we want to make action $open$ urgent in the conveyor

[3]Note that an urgent action in CIF does not have priority over other urgent actions.

model. This can be achieved with the aid of the urgency operator as shown in Equation **??**.

$$\upsilon_{open}(\gamma_{close}(\gamma_{open}(p_{Tank})) \parallel \gamma_{close}(\gamma_{open}(p_{Conveyor}))) \tag{4}$$

In this model, if action *open* is enabled in both components (action *open* is synchronizing), then time cannot longer pass and an action has to be executed (although not necessarily *open*). In the model under consideration, the execution of action *open* occurs at time Δ, since it is the *earliest* time at which this action is enabled.

Note that if we would have used two urgency operators to enclose each component of the bottle filling line separately, as shown in Equation **??**, the model would have deadlocked, since action *open* is enabled at time 0 in the tank, and therefore time cannot pass. But this action is disabled in the conveyor, and since it must be executed synchronously no action can be executed either.

$$\upsilon_{open}(\gamma_{close}(\gamma_{open}(p_{Tank}))) \parallel \upsilon_{open}(\gamma_{close}(\gamma_{open}(p_{Conveyor}))) \tag{5}$$

4.7. Other operators

In this section we explain the informal semantics of the remaining CIF operators. Their semantics was introduced in previous works [**? ?**], and therefore they do not require a separate section for each one of them. Section **??** provides the formal semantics of these operators.

The initialization operator allows to define the initial conditions of models in a compositional way. Given a composition p and a predicate u, composition $u \gg p$ can start executing in initial valuations that satisfy u. For instance, we could have set the initial volume of the tank in the following way:

$$(V_T = 10) \gg (\gamma_{close}(\gamma_{open}(p_{Tank})) \parallel \gamma_{close}(\gamma_{open}(p_{Conveyor}))).$$

Actions and variables can be made local by means of the scoping operators. Given a composition p and an action a, composition $|[_{\mathrm{A}}\, a :: p\ ]|$ declares action a as local, which means that a is not visible outside this composition, and as a consequence it cannot be synchronized with other actions in a parallel context. Similarly, given a composition p, expressions e_0 and e_1, and a variable x, composition $|[_{\mathrm{V}}\, x = e_0, \dot{x} = e_1 :: p\ ]|$ declares variables x and $\dot{x}$ as local, which means that the local values of these variables are not visible outside this scope. Expressions e_0 and e_1 define the initial values of the local variables. These expressions can be $\perp$, which indicates that the variables are uninitialized and therefore they can have any value in their domain.

5. Formal semantics

In this section, we present the structured operational semantics (SOS) of CIF. Formal semantics has several advantages since it gives a precise meaning to CIF models, it provides a reference against which implementations can be judged, and it enables the formal definition and proof of semantics-preserving model transformations.

Instead of taking the traditional approach for defining the semantics of hybrid automata [**?**], we have chosen the SOS approach since it is better suited for guiding the implementation process [**?**], and it allows us to use standard formats for proving congruence [**?**].

The formal semantics described here associates, by means of inference rules, a hybrid transition system with every CIF composition. The following sections present the inference rules for constructing the three kinds of transition relations for a given composition. The SOS rules are presented in the same order as the operators are introduced in the abstract grammar of Table **??**.

5.1. Automata

Rule **??** models the state change caused by the execution of an action, which can be triggered in a state with valuation σ, only when there is a location v, and an edge of the form $(v, g, a, (W, r), v')$, such that the initialization predicate of v ($\mathrm{init}(v)$), the guard g, and the invariant of v ($\mathrm{inv}(v)$) are satisfied in σ; and it is possible to find a new valuation σ' such that the variables in the set $(X \cup \mathrm{var}_C) \setminus W$ do not change in σ', the update predicate r is satisfied[4] in $\sigma'^{+} \cup \sigma$, and the invariant of the new location is satisfied in σ'. In Rule **??**, id_v denotes the function that returns true only if the location equals v. More specifically, $\mathrm{id}_v \in V \to \mathbb{B}$, and $\mathrm{id}_v(w) \triangleq\ v \equiv w$, where $\equiv$ denotes syntactic equivalence.

$$\frac{\begin{array}{c}(v, g, a, (W, r), v') \in E, \sigma \models \mathrm{init}(v), \sigma \models g, \sigma \models \mathrm{inv}(v), \sigma' \models \mathrm{inv}(v'),\\ \sigma'^{+} \cup \sigma \models r, \sigma \restriction_{(X \cup \mathrm{var}_C)\setminus W} = \sigma' \restriction_{(X \cup \mathrm{var}_C)\setminus W}\end{array}}{\begin{array}{c}((V, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, E, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype}), \sigma) \xrightarrow{a, a \in \mathrm{act}_S, X}\\ ((V, \mathrm{id}_{v'}, \mathrm{inv}, \mathrm{tcp}, E, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype}), \sigma')\end{array}} \; 1$$

The previous rule shows how the initial predicate function is used as a program counter, which keeps track of the actions that are executed. Note that a unique active location is chosen after performing action a. As we will see later on, the same happens when performing a time delay, or an environment transition. In

[4]Remember that $\sigma'^{+} = \{(x^{+}, v) \mid (x, v) \in \sigma'\}$ as defined in Section **??**.

an action transition we do not want control variables to jump arbitrarily. Control variables can be declared in the automaton (var_C), or in the control variable operator, in which case these declared variables will be included in the set X. This is why we demand condition $\sigma \restriction_{(X \cup \mathrm{var}_C)\setminus W} = \sigma' \restriction_{(X \cup \mathrm{var}_C)\setminus W}$ to hold. Condition ${\sigma'}^{+} \cup \sigma \models r$ is needed (instead of $\sigma' \models r$) because r may refer to the values of the variables before the action, which are in σ; and also to the new values of the variables, which are in ${\sigma'}^{+}$.

Rule **??** models the passage of time for an automaton. Time can pass for t time units if the invariant associated with the active locations is satisfied in each point in $[0, t]$, the time can progress predicate is satisfied in $[0, t)$, and the dynamic type constraints specified by dtype are satisfied. Note that according to this rule, a time transition can select in which of the possible initial states the automaton begins its execution. This corresponds to the theory of hybrid automata (e.g. see [**?**]).

$$\frac{\begin{array}{c}\mathrm{dom}(\rho) = [0, t], 0 < t, \mathrm{dom}(\rho) = \mathrm{dom}(\theta), \rho(0) \models \mathrm{init}(v),\\ \langle \forall s : s \in \mathrm{dom}(\theta) : \theta(s) = \{a | (v, g, a, u, v') \in E \wedge \rho(s) \models g\}\rangle,\\ \langle \forall s : s \in [0, t] : \rho(s) \models \mathrm{inv}(v)\rangle, \langle \forall s : s \in [0, t) : \rho(s) \models \mathrm{tcp}(v)\rangle,\\ \langle \forall x : x \in \mathrm{dom}(\mathrm{dtype}) : (\rho_x, \rho_{\dot{x}}) \in \mathrm{dtype}(x)\rangle\end{array}}{\begin{array}{c}((V, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, E, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype}), \rho(0)) \overset{\rho, \mathrm{act}_S, \theta}{\longmapsto}\\ ((V, \mathrm{id}_v, \mathrm{inv}, \mathrm{tcp}, E, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype}), \rho(t))\end{array}} \; 2$$

Finally, Rule **??** states that an automaton can perform an environment transition in an initial valuation σ, if there is an active location v such that σ is consistent with the invariant of v, and it can end in any valuation σ' that satisfies the same invariant, and where the variables controlled by the automaton remain unchanged.

$$\frac{\sigma \models \mathrm{init}(v),\ \sigma \models \mathrm{inv}(v),\ \sigma' \models \mathrm{inv}(v),\ \sigma \restriction_{\mathrm{var}_C} = \sigma' \restriction_{\mathrm{var}_C}}{\begin{array}{c}((V, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, E, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype}), \sigma) \overset{\mathrm{act}_S}{\dashrightarrow}\\ ((V, \mathrm{id}_v, \mathrm{inv}, \mathrm{tcp}, E, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype}), \sigma')\end{array}} \; 3$$

5.2. Parallel composition

Rule **??** states that two synchronizing actions with the same label can execute in parallel only if they share the same initial and final valuation, and if the action is synchronizing in both compositions. The set of control variables X, is propagated from the conclusions to the premises since the control variables in the scope of a parallel composition are shared by both partners. The resulting action transition is also synchronizing, which allows action a to synchronize with more than two compositions.

$$\frac{(p,\sigma) \xrightarrow{a,\text{true},X} (p',\sigma'), (q,\sigma) \xrightarrow{a,\text{true},X} (q',\sigma')}{(p \parallel q,\sigma) \xrightarrow{a,\text{true},X} (p' \parallel q',\sigma')}\ 4$$

Rules **??** and **??** model interleaving behavior of two compositions when executed in parallel. In these rules, an action can be performed in one of the components only if the initial and final valuations are consistent with the other composition, and if this action is not synchronizing in the other component, which is expressed by condition $a \notin A$. In Rule **??**, the environment transition $(q,\sigma) \overset{A}{\dashrightarrow} (q',\sigma')$ is used to obtain the set of synchronizing action labels in composition q, to ensure that the initial valuation σ is consistent with the active invariants and initialization conditions of q, to select an initial location (in case there is more than one in q), and to remove any initialization operators from q[5]. Note that q', and not q, is placed in the final state in the conclusion; this avoids occurrences of the initialization operator in the resulting term and stores changes in local variables (if any).

$$\frac{(p,\sigma) \xrightarrow{a,b,X} (p',\sigma'),\ (q,\sigma) \overset{A}{\dashrightarrow} (q',\sigma'),\ a \notin A}{(p \parallel q,\sigma) \xrightarrow{a,b,X} (p' \parallel q',\sigma')}\ 5$$

$$\frac{(q,\sigma) \xrightarrow{a,b,X} (q',\sigma'),\ (p,\sigma) \overset{A}{\dashrightarrow} (p',\sigma'),\ a \notin A}{(p \parallel q,\sigma) \xrightarrow{a,b,X} (p' \parallel q',\sigma')}\ 6$$

Rule **??** models the fact that if two compositions are put in parallel, time can pass only if allowed by both partners. As can be seen in this rule, the set of enabled actions in the parallel composition at any point in time during the delay depends both on the set of enabled actions and the set of synchronizing actions in each component individually. In the expression $(\theta_p \cap \theta_q) \cup (\theta_p \setminus A_q) \cup (\theta_q \setminus A_p)$ in the conclusion, the constants A_p and A_q, and the operators $\cap$ and $\setminus$ must be lifted accordingly to match the types. We avoid doing so here to keep the notation simple.

$$\frac{(p,\sigma) \overset{\rho,A_p,\theta_p}{\longmapsto} (p',\sigma'),\ (q,\sigma) \overset{\rho,A_q,\theta_q}{\longmapsto} (q',\sigma')}{(p \parallel q,\sigma) \overset{\rho,A_p \cup A_q,(\theta_p \cap \theta_q) \cup (\theta_p \setminus A_q) \cup (\theta_q \setminus A_p)}{\longmapsto} (p' \parallel q',\sigma')}\ 7$$

The construction of the guard trajectory in the conclusion deserves some explanation. According to the description of the time transition given in Section **??**,

[5]Removing initialization operators is important because we want them to influence only the initial behavior of the system.

the guard trajectory in the conclusion should model the set of enabled actions at a given time in the parallel composition $p \parallel q$. Let A_i be the set of synchronizing actions of composition i, $i \in \{p, q\}$, and $\theta_i(s)$ the set of enabled actions of i at time s, then an action a is enabled at time s in $p \parallel q$ if at least one of the following conditions holds:

1. Action a is enabled in both components, which can be expressed as $a \in \theta_p(s) \cap \theta_q(s)$. A simple case analysis will convince the reader that when an action is enabled in both components, then it is enabled in the parallel composition of these, irrespective of whether a is synchronizing.
2. Action a is enabled in p, and is not synchronizing in q, which can be expressed as $a \in \theta_p(s) \setminus A_q$.
3. Symmetrically, $a \in \theta_q(s) \setminus A_p$.

Rule **??** defines the environment transition behavior for parallel composition. The resulting set of synchronizing actions is the union of the synchronizing actions of p and q, and the end valuation of all transitions must match.

$$\frac{(p,\sigma) \overset{A_p}{\dashrightarrow} (p',\sigma'),\ (q,\sigma) \overset{A_q}{\dashrightarrow} (q',\sigma')}{(p \parallel q,\sigma) \overset{A_p \cup A_q}{\dashrightarrow} (p' \parallel q',\sigma')}\ 8$$

5.3. Control variable operator

In Rule **??**, variable x is added to the set of control variables of the transition in the premise. In this way, x can only be changed by an automaton if this change is explicitly specified (see Rule **??**).

$$\frac{(p,\sigma) \xrightarrow{a,b,X \cup \{x\}} (p',\sigma')}{(\mathrm{ctrl}_x(p),\sigma) \xrightarrow{a,b,X} (\mathrm{ctrl}_x(p'),\sigma')}\ 9$$

The operator has no effect in time transitions, as expressed by Rule **??**. Rule **??** allows an environment transition only if the control variable is not changed in the premise. Thus, when used in interleaving parallel composition, this will ensure that the value of a control variable will not be changed by a parallel process outside the scope of the operator.

$$\frac{(p,\sigma) \overset{\rho,A,\theta}{\longmapsto} (p',\sigma')}{(\mathrm{ctrl}_x(p),\sigma) \overset{\rho,A,\theta}{\longmapsto} (\mathrm{ctrl}_x(p'),\sigma')}\ 10 \qquad \frac{(p,\sigma) \overset{A}{\dashrightarrow} (p',\sigma'),\ \sigma(x) = \sigma'(x)}{(\mathrm{ctrl}_x(p),\sigma) \overset{A}{\dashrightarrow} (\mathrm{ctrl}_x(p),\sigma')}\ 11$$

5.4. Urgency operator

Rule **??** specifies that the urgency operator restricts the time behavior of a composition in such a way that time can pass for as long as no urgent action is enabled.

$$\frac{\mathrm{dom}(\rho) = [0,t], (p,\sigma) \overset{\rho,A,\theta}{\longmapsto} (p',\sigma'),\ \langle \forall s : s \in [0,t) : a \notin \theta(s)\rangle}{(\upsilon_a(p),\sigma) \overset{\rho,A,\theta}{\longmapsto} (\upsilon_a(p'),\sigma')} \ 12$$

The urgency operator affects only the time transition rules. Action and environment transitions remain unchanged as expressed in Rules **??** and **??** (although action behavior can be influenced by the urgency operator, since it can prevent certain guarded actions from taking place if urgent actions are enabled).

$$\frac{(p,\sigma) \xrightarrow{a',b,X} (p',\sigma')}{(\upsilon_a(p),\sigma) \xrightarrow{a',b,X} (\upsilon_a(p'),\sigma')} \ 13 \qquad \frac{(p,\sigma) \overset{A}{\dashrightarrow} (p',\sigma')}{(\upsilon_a(p),\sigma) \overset{A}{\dashrightarrow} (\upsilon_a(p'),\sigma')} \ 14$$

5.5. Dynamic type operator

Rule **??** specifies that, for time transitions, the dynamic type operator ensures that the trajectories for the variable x and the dotted version of the variable $\dot{x}$ are restricted to the behavior that is specified by the set of pairs of solution functions G.

$$\frac{(p,\sigma) \overset{\rho,A,\theta}{\longmapsto} (p',\sigma'),\ (\rho_x, \rho_{\dot{x}}) \in G}{(\mathrm{D}_{x:G}(p),\sigma) \overset{\rho,A,\theta}{\longmapsto} (\mathrm{D}_{x:G}(p'),\sigma')} \ 15$$

The dynamic type operator has no effect on action or environment transitions as expressed by Rules **??** and **??**.

$$\frac{(p,\sigma) \xrightarrow{a,b,X} (p',\sigma')}{(\mathrm{D}_{x:G}(p),\sigma) \xrightarrow{a,b,X} (\mathrm{D}_{x:G}(p'),\sigma')} \ 16 \qquad \frac{(p,\sigma) \overset{A}{\dashrightarrow} (p',\sigma')}{(\mathrm{D}_{x:G}(p),\sigma) \overset{A}{\dashrightarrow} (\mathrm{D}_{x:G}(p'),\sigma')} \ 17$$

5.6. Initialization operator

Rules **??**, **??**, and **??** formalize the fact that the initialization operator allows transitions only for those initial valuations that satisfy the initialization predicate.

$$\frac{(p,\sigma) \overset{A}{\dashrightarrow} (p',\sigma'), \sigma \models u}{(u \gg p,\sigma) \overset{A}{\dashrightarrow} (p',\sigma')} \ 18 \qquad \frac{(p,\sigma) \xrightarrow{a,b,X} (p',\sigma'), \sigma \models u}{(u \gg p,\sigma) \xrightarrow{a,b,X} (p',\sigma')} \ 19$$

$$\frac{(p,\sigma) \overset{\rho,A,\theta}{\longmapsto} (p',\sigma'), \sigma \models u}{(u \gg p,\sigma) \overset{\rho,A,\theta}{\longmapsto} (p',\sigma')} \ 20$$

5.7. Synchronization operator

The term $\gamma_a(p)$ defines the basic action label a as synchronizing. Synchronization occurs only in the context of parallel composition, thus the composition $\gamma_a(p)$ must be placed in parallel with another component to observe the effect of the synchronization operator.

The above is formalized in the SOS rules by altering the edges of the transitions, as shown in Rules **??**, **??**, and **??**. More specifically, for time behavior (Rule **??**), the synchronization operator only changes the set of synchronizing actions in the labels of time transitions. For action transitions (Rule **??**) the action a is made synchronizing if the action label coincides with a, and for environment transitions (Rule **??**) a is added to the set of synchronizing actions given by the premise.

$$\frac{(p,\sigma) \overset{\rho,A,\theta}{\longmapsto} (p',\sigma')}{(\gamma_a(p),\sigma) \overset{\rho,A\cup\{a\},\theta}{\longmapsto} (\gamma_a(p'),\sigma')} \; 21$$

$$\frac{(p,\sigma) \xrightarrow{a',b,X} (p',\sigma')}{(\gamma_a(p),\sigma) \xrightarrow{a',b\vee a'=a,X} (\gamma_a(p'),\sigma')} \; 22 \qquad \frac{(p,\sigma) \overset{A}{\dashrightarrow} (p',\sigma')}{(\gamma_a(p),\sigma) \overset{A\cup\{a\}}{\dashrightarrow} (\gamma_a(p'),\sigma')} \; 23$$

5.8. Variable scope operator

The variable scope operator (also called variable hiding) introduces a local variable and its dotted version in a composition. These variables are invisible outside of the scope, and, conversely, global variables with the same name cannot be modified inside the scope.

Rule **??** is similar to the variable scope rule presented in [**?**]. In its premise, the variable being declared is removed from the set of control variables, because if $x \in X$ then x refers to a global variable. Predicate $(\sigma \models (e_i = d_i)) \vee e_i = \bot$ models the fact that the value of expression e in the valuation σ is d_i, unless $e_i = \bot$, in which case d_i can be any value. Condition $x \in X \Rightarrow \sigma(x) = \sigma'(x)$ in rule **??** is used since in an action transition $(p,\sigma) \xrightarrow{a,b,X} (p',\sigma')$, it must be the case that $\sigma \restriction_{(X\backslash W)} = \sigma' \restriction_{(X\backslash W)}$, where W is the set of all the variables that are changed by action a; and since all ocurrences of variable x in the set of jumping variables in the automaton refers to the local variable, the global variable x cannot be a jumping variable of the action a. Therefore, if it is a control variable then it cannot change its value.

$$\frac{\begin{array}{c}(\sigma \models (e_0 = d_0)) \vee e_0 = \bot, (\sigma \models (e_1 = d_1)) \vee e_1 = \bot, x \in X \Rightarrow \sigma(x) = \sigma'(x) \\ (p, \{x \mapsto d_0, \dot{x} \mapsto d_1\} \succ \sigma) \xrightarrow{a,b,X\backslash\{x\}} (p', \{x \mapsto d'_0, \dot{x} \mapsto d'_1\} \succ \sigma')\end{array}}{(|[_{\mathrm{V}}\, x = e_0, \dot{x} = e_1 :: p\,]|, \sigma) \xrightarrow{a,b,X} (|[_{\mathrm{V}}\, x = d'_0, \dot{x} = d'_1 :: p'\,]|, \sigma')} \; 24$$

Rule **??** is similar to the action transition rule for variable scope. For this rule we make use of the *trajectory overwriting* operator, which behaves in a similar way as the function overwriting operator, and is defined below.

Definition 9 (Trajectory overwriting)**.** *Given a time point t, trajectory $\rho \in [0,t] \to \mathcal{V} \to \Lambda$, a function $f \in [0,t] \to \Lambda$, and a variable x, function $(\{x \mapsto f\} \succ \rho) \in [0,t] \to \mathcal{V} \to \Lambda$ is defined as follows:*

$$(\{x \mapsto f\} \succ \rho)(s)(y) = \begin{cases} f(s) & \text{if } x = y \\ \rho(s)(y) & \text{if } x \neq y \end{cases}$$

for all $s \in [0,t]$, for all $y \in \mathcal{V}$.

Using the trajectory overwriting operator, we can define the time rule for the variable scope operator.

$$\frac{(\sigma \models e_0 = d_0) \vee e_0 = \bot, (\sigma \models e_1 = d_1) \vee e_1 = \bot, \mathrm{dom}(\rho) = [0,t] \quad (p, \{x \mapsto d_0, \dot{x} \mapsto d_1\} \succ \sigma) \overset{\{x \mapsto f_0, \dot{x} \mapsto f_1\} \succ \rho, A, \theta}{\longmapsto} (p', \{x \mapsto d_0', \dot{x} \mapsto d_1'\} \succ \sigma')}{(\|[_{\mathrm{V}}\, x = e_0, \dot{x} = e_1 :: p\,]\|, \rho(0)) \overset{\rho, A, \theta}{\longmapsto} (\|[_{\mathrm{V}}\, x = d_0', \dot{x} = d_1' :: p'\,]\|, \rho(t))} \; 25$$

In this rule, the initial valuation $\rho(0)$ is used in the conclusion since we must ensure that for all time transitions $(p, \sigma) \overset{\rho, A, \theta}{\longmapsto} (p', \sigma')$ it is the case that $\sigma = \rho(0)$. The same holds for the final valuation in the conclusion.

Finally Rule **??**, which is similar to Rule **??** for action transitions, specifies the environment behavior for the variable scope operator.

$$\frac{(\sigma \models e_0 = d_0) \vee e_0 = \bot, (\sigma \models e_1 = d_1) \vee e_1 = \bot, \quad (p, \{x \mapsto d_0, \dot{x} \mapsto d_1\} \succ \sigma) \overset{A}{\dashrightarrow} (p', \{x \mapsto d_0', \dot{x} \mapsto d_1'\} \succ \sigma')}{(\|[_{\mathrm{V}}\, x = e_0, \dot{x} = e_1 :: p\,]\|, \sigma) \overset{A}{\dashrightarrow} (\|[_{\mathrm{V}}\, x = d_0', \dot{x} = d_1' :: p'\,]\|, \sigma')} \; 26$$

5.9. *Action scope operator*

The action scope operator introduces local actions in an automaton. These actions are visible outside the scope as internal τ actions, and therefore cannot be synchronized with global actions (note that τ cannot be made synchronizing). Rules **??** and **??** model the action behavior of this operator, where the name of the local action is replaced (in the arrows of the action transitions) by τ, and synchronization is prevented.

$$\frac{(p, \sigma) \xrightarrow{a, b, X} (p', \sigma')}{(\|[_{\mathrm{A}}\, a :: p\,]\|, \sigma) \xrightarrow{\tau, \mathrm{false}, X} (\|[_{\mathrm{A}}\, a :: p'\,]\|, \sigma')} \; 27$$

$$\frac{(p,\sigma) \xrightarrow{a',b,X} (p',\sigma'),\ a' \neq a}{(|[_{\mathrm{A}}\, a :: p\ ]|, \sigma) \xrightarrow{a',b,X} (|[_{\mathrm{A}}\, a :: p'\ ]|, \sigma')}\ 28$$

For time transitions (Rule **??**), the operator affects the set of synchronizing actions as well as the guard trajectory. Since a is not visible outside its scope, if a is declared as urgent in an outer scope this should not affect the time behavior of, for instance, $\upsilon_a(|[_{\mathrm{A}}\, a :: p\ ]|)$. Thus we must remove a from the guard trajectory. As in the previous rules, we have overloaded the symbols $\{a\}$ and $\setminus$ used in the guard trajectory in such a way they the match the types.

$$\frac{(p,\sigma) \overset{\rho,A,\theta}{\longmapsto} (p',\sigma')}{(|[_{\mathrm{A}}\, a :: p\ ]|, \sigma) \overset{\rho,A\setminus\{a\},\theta\setminus\{a\}}{\longmapsto} (|[_{\mathrm{A}}\, a :: p'\ ]|, \sigma')}\ 29$$

In Rule **??**, for environment transitions, the operator removes the local action from the set of synchronizing actions, since it cannot synchronize outside its scope.

$$\frac{(p,\sigma) \overset{A}{\dashrightarrow} (p',\sigma')}{(|[_{\mathrm{A}}\, a :: p\ ]|, \sigma) \overset{A\setminus\{a\}}{\dashrightarrow} (|[_{\mathrm{A}}\, a :: p'\ ]|, \sigma')}\ 30$$

6. CIF as an interchange format

This section discusses CIF as an interchange format. In Section **??** we illustrate the relation that exists between CIF and some of the most important formalisms for modeling timed and hybrid systems. For this, we show how ubiquitous concepts in some of the most prominent timed and hybrid formalisms can be represented in CIF. To the best of our knowledge, the combination of concepts that CIF features is not present in any other compositional formalism. The blend of different concepts in CIF gives as a result a formalism where models may not be directly expressed in the target formalisms, such as Uppaal. In Section **??** we explain how linearization can help to overcome this problem. A valid question is whether the formalisms discussed in this section cannot be used as interchange format instead of CIF. To answer this, in Section **??** we highlight features of CIF that are not existent in these formalism.

For more details about transformations between CIF and formalisms and tools such as Uppaal, SpaceEx, gPROMS, Modelica, Sequential Function Charts, and Matlab/Simulink, we refer the interested reader to the material cited in Section 1.3.

6.1. Direct mapping of concepts from other formalisms to CIF

6.1.1. Uppaal

Uppaal [**?**] is a formalism for modeling timed systems using timed automata. Its associated tool [**?**] is very well known in the area of verification and analysis of timed systems.

As in CIF, a Uppaal automaton consists of locations to specify the different computational states of a system, and transitions between locations to model the discrete behavior. Figure **??** shows two Uppaal automata, having four locations and three transitions. The passage of time is captured by clocks, which can be reset in transitions. In Figure **??** there are two clock variables, x and y. Each location contains an invariant predicate that restricts the passage of time: time can pass for as long as the invariant is satisfied. In Figure **??**, time delays in location l_0 are possible for as long as $x \leq 10$. Transitions between locations can have guards, a synchronization label (used as a channel), and an variable update. In Figure **??** the edge from l_0 to l_1 can be triggered if $5 \leq x$. Similarly, executing the action a? in automaton *Beta* causes the clock variable y to be reset to 0. These two edges can only be executed synchronously since the two automata communicate via channel a. Automata can be composed in parallel, forming what is called a *network of automata*.

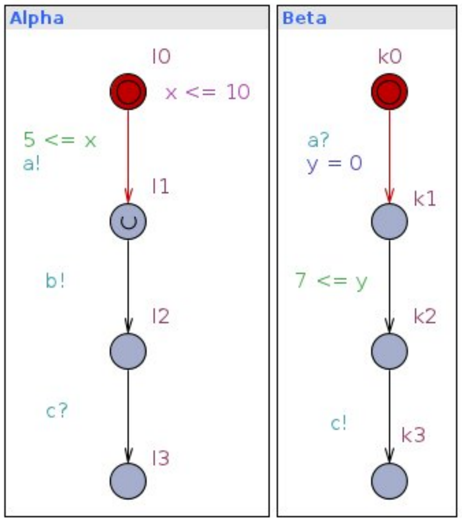
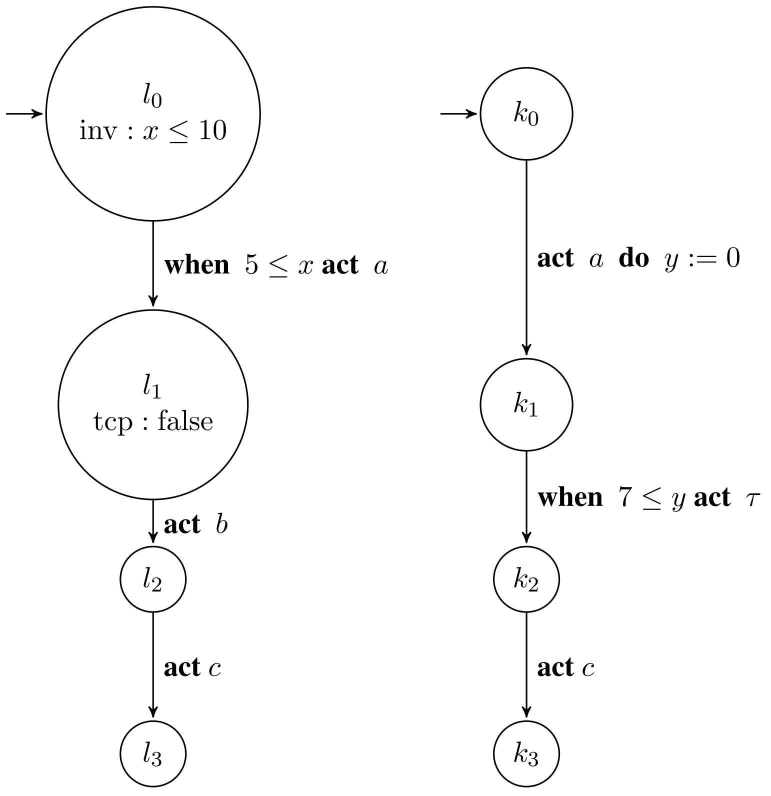


Figure 22: A network of two automata in Uppaal.

Additionally, locations can be marked as *urgent*, which means that time cannot pass when the system is in that location. In Figure **??** location l_1 is marked as urgent, and therefore no time delays are possible until the system leaves it. Channels can also be declared as urgent. In this case, time cannot pass if a synchronization in that channel is possible. For the example we are considering in this section, if automaton *Alpha* is in location l_2 and *Beta* is in location k_1, then time can pass until the edge going from k_1 to k_2 is triggered. After that, synchronization on channel c is possible, and since this channel is urgent, delays are no longer possible.

Equation **??** shows the CIF translation of the Uppaal model presented below, where automata α and β are depicted in Figure **??**.

$$\upsilon_c(\gamma_{a,c,b}(\alpha) \parallel \gamma_{a,c}(\beta)) \tag{6}$$



(a) Automaton α. (b) Automaton β.

Figure 23: Uppaal automata translated to CIF.

6.1.2. SpaceEx

SpaceEx is one of the most sophisticated tools for reachability analysis of hybrid automata [**?**]. SpaceEx models consist of one or more hybrid automata, which interact in parallel by means of synchronizing actions and shared variables. Figure **??** shows a SpaceEx model of a bouncing ball. Its semantics is similar to the one of CIF. Locations are depicted as rectangles with rounded corners to be consistent with the notation used in the SpaceEx related publications. Variable h represents the height of the ball, v its velocity, g is the gravitational constant, and d is the dampening factor. In the state $physics$, the first predicate $0 \leq h$ represents the *invariant*, whereas the second predicate $h' == v \wedge v' == g$ is used to represent the *flow conditions* that govern the continuous evolution of the variables during time delays. The automaton contains only one edge, exiting and entering the same location. The first line, hop, represents the action that is executed; predicate $h \leq 0 \wedge v < 0$ represents the guard; and the assignment $v := -d * v$ determines how the value of variables get updated after performing the action.

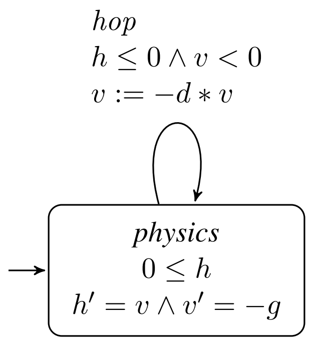


Figure 24: SpaceEx model of a bouncing ball.

SpaceEx supports the notion of controlled variables. In the automaton of Figure **??** we assume variable h is controlled, which means that only that automaton can change its value.

Flows and invariants in SpaceEx are readily translated to CIF invariants. Guards, actions, and updates in the edges of the SpaceEx automata can be represented using the corresponding CIF concepts. Finally, controlled variables can be expressed also in CIF by means of the control variable operator. Figure **??** shows the model of the bouncing ball in CIF.

In SpaceEx, actions can be *connected*, and as a result, all the connected actions must be executed synchronously in the model. Automata can read the controlled variables in other automata. Figures **??** and **??** show two additional automata respectively: a *hop counter*, which keeps track of the number of times the ball hits the ground; and a *controller* automaton, which re-initializes the ball velocity to I when it is bouncing below a certain threshold MH (minimum height) for a given period of time T. Action hop of automata *bouncing ball* and *hop counter* are

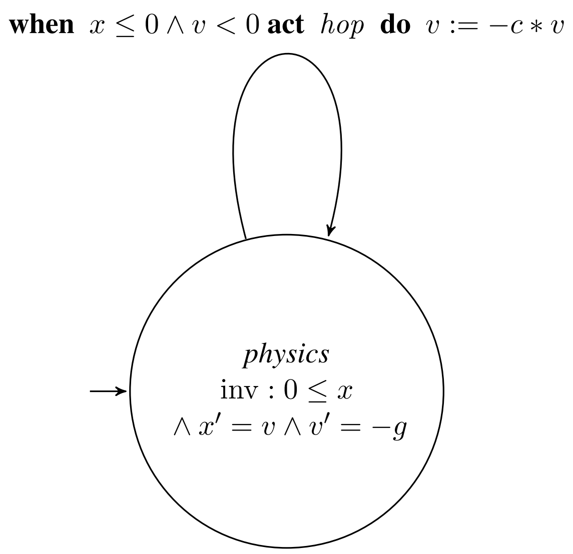

when $x \leq 0 \wedge v < 0$ **act** hop **do** $v := -c * v$
physics
inv : $0 \leq x$
$\wedge\, x' = v \wedge v' = -g$

Figure 25: CIF model of a bouncing ball (α_{bball}).

connected, and therefore they synchronize. Automaton *controller* can only read variable *x*, whereas the velocity, represented by variable v, is controlled at the top level, which means that all the automata can change its value.

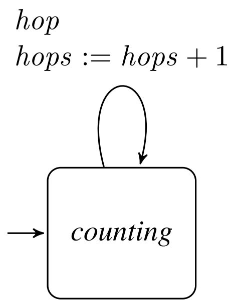


Figure 26: SpaceEx model of the hop counter

In CIF the synchronization of the connected actions is achieved by means of the synchronizing action operator. The control information about variables is specified using the control variable operator. Equation (**??**) shows this, where automata α_{hop} and $\alpha_{controller}$ are the CIF translation of the automata *hop counter* and *controller*, respectively. We omit is definition here since it is trivial, as can be observed from

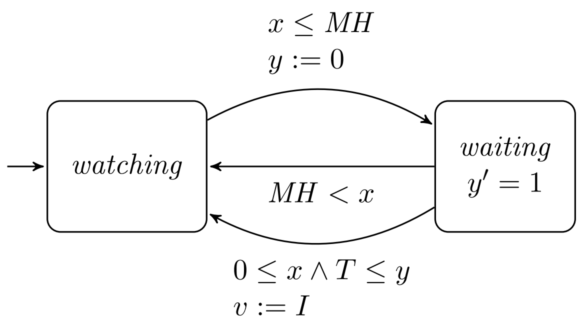


Figure 27: SpaceEx model of the ball controller

the bouncing ball translation.

$$\mathrm{ctrl}_{v,g,I,T,MH}(\gamma_{hop}(\mathrm{ctrl}_x(\alpha_{bball})) \parallel \gamma_{hop}(\mathrm{ctrl}_{hops}(\alpha_{hop})) \parallel \alpha_{controller}) \qquad (7)$$

6.1.3. gPROMS

gPROMS is an equation-based modeling language for describing and simulating complex, controlled, hybrid systems, which focuses on the process industry. It is an structured language that features a clear division between continuous dynamic specifications (called "models"), and discrete behavior specifications (called "processes").

Listing **??** shows the gPROMS specification corresponding to the continuous behavior of a bouncing ball. It contains two equations separated by a semi-colon. The first equation `$h=v`, bounds the derivative of variable `h`, represented using the dollar sign, to variable `v`. This equation has to hold at all times during the simulation. The second equation is a case equation. Variable `ballPosition` behaves as a location or program counter. If the variable is set to `air`, then equation `$v = -g` is active. If the variables is set to `ground`, then equation `$v = 0` is active. Changes to the active equations happen when the switching conditions specified by the **SWITCH TO** keywords become true.

Listing 1: Equations of the bouncing ball model.

```
$h=v;
CASE ballPosition OF
    WHEN air:
        $v=-g;
        SWITCH TO ground IF h<=0.0 AND v<=0.0;
    WHEN ground:
        $v=0.0;
        SWITCH TO air IF NOT h<=0.0 AND v<=0.0;
END
```

The CIF model corresponding to the bouncing ball equations consist of the parallel composition of two automata. The first equation is translated to automaton that has only one location and predicate $\dot{h} = v$ as invariant. The case equation is translated into the automaton shown in Figure **??**. The branches of the case statement are translated into locations in the automaton. The invariants of each location are the equations of the case branches. Locations are connected with edges having as guards the switching conditions. To reflect the fact that switching is urgent (*i.e.* no time can pass) the appropriate tcp predicates are added.

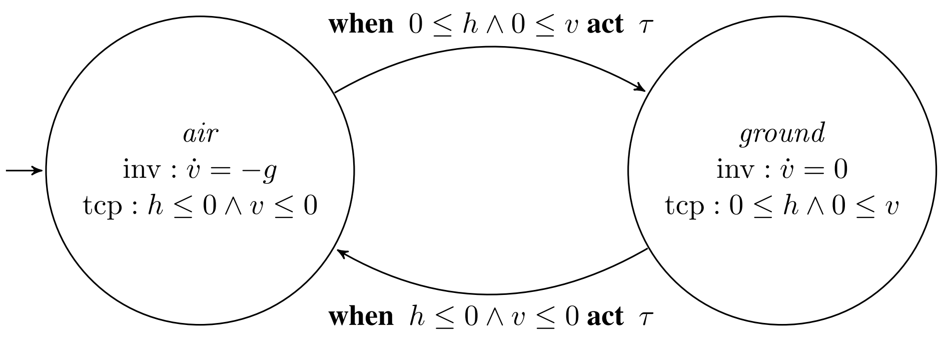

$\mathbf{when}\ 0 \leq h \wedge 0 \leq v\ \mathbf{act}\ \tau$
air
$\text{inv} : \dot{v} = -g$
$\text{tcp} : h \leq 0 \wedge v \leq 0$
$ground$
$\text{inv} : \dot{v} = 0$
$\text{tcp} : 0 \leq h \wedge 0 \leq v$
$\mathbf{when}\ h \leq 0 \wedge v \leq 0\ \mathbf{act}\ \tau$

Figure 28: gPROMS case equation translated to CIF.

The discrete behavior of the bouncing ball is modeled using gPROMS process. Listing **??** shows this specification in gPROMS. The **CONTINUE UNTIL** command waits until condition `0 <= h` becomes true, and then it proceeds with the next statement. **REINITIAL** can be regarded as ordinary assignments. In gPROMS all discrete commands must be executed when they are enabled.

Listing 2: Discrete part of the bouncing ball model.

```
WHILE TRUE DO
    SEQUENCE
        CONTINUE UNTIL h<0.0
        REINITIAL
            v
        WITH
            v=-d*OLD(v);
        END
        REINITIAL
```

```
                h
            WITH
                h=0.0;
            END
            RESET
                n:=OLD(n)+1.0;
            END
        END
END
```

Figure **??** shows the CIF automaton corresponding to the gPROMS specification of Listing **??**. The `CONTINUE UNTIL` is translated into a location whose tcp predicate is the closed negation of the continuation condition. An edge with this condition as guard connects this location with the location corresponding to the next instruction. Assignments are translated into CIF assignments, and the tcp predicates of the remaining locations are set to false to reflect the fact that these sentences cannot be delayed.

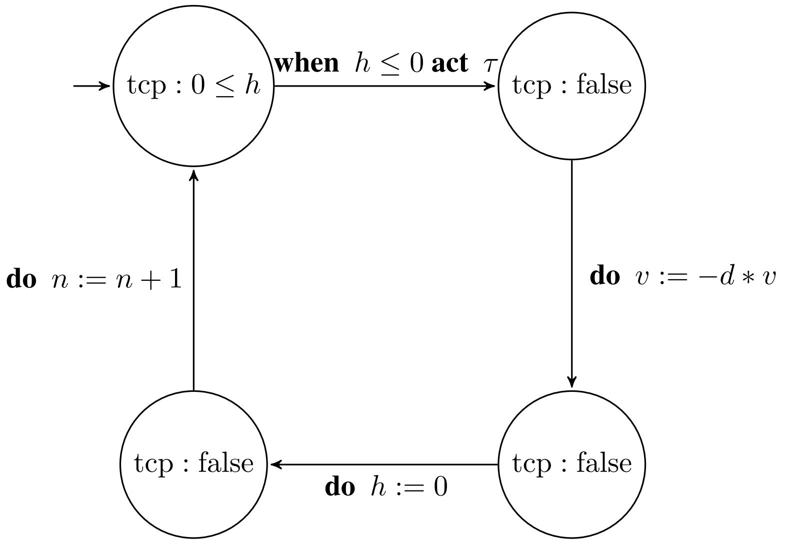


Figure 29: gPROMS process translated to CIF.

Additional elements of gPROMS that are straightforwardly supported by CIF include also parallel composition, if statements, among others.

6.1.4. Chi

Chi is a formalism for modeling and simulation of hybrid systems [**?**]. It is based on process algebra, and it integrates concepts from dynamics and control theory. Equation **??** shows a Chi model of a tank controller. It consists of a discrete variable n, which represents the status of the valve (1 means open, 0 closed), a continuous variable V representing the tank volume, and two variables Q_i and Q_o representing the incoming and outgoing flow from the tank, respectively. The model consists of a variables declaration part, followed by an initial assignment of values to the model variables, and a model specification part. The latter component consists of the parallel composition of an equation of the form:

$$\dot{V} = Q_{\mathrm{i}} - Q_{\mathrm{o}},\ Q_{\mathrm{i}} = n \cdot 5,\ Q_{\mathrm{o}} = \sqrt{V} \tag{8}$$

together with an infinite repetition (specified by the operator $*(\_)$) of two sequential guarded assignments. In equations, a comma stands for the conjunction operator.

$$\begin{array}{ll} \langle & \mathbf{disc}\ n, \mathbf{cont}\ V, \mathbf{alg}\ Q_{\mathrm{i}}, Q_{\mathrm{o}} \\ , & n = 0, V = 10 \\ | & \dot{V} = Q_{\mathrm{i}} - Q_{\mathrm{o}} \\ , & Q_{\mathrm{i}} = n \cdot 5 \\ , & Q_{\mathrm{o}} = \sqrt{V} \\ \| & *(V \leq 2 \rightarrow n := 1; V \geq 10 \rightarrow n := 0) \\ \rangle & \end{array} \tag{9}$$

Elements such as equations, guarded action updates, sequential composition, alternative composition, while loops, and tail recursion can be easily translated to CIF. Figure **??** shows the CIF model corresponding to the discrete part of the Chi model of Equation (**??**). The continuous part is realized though an automaton having one initial location, and Equation (**??**) as invariant, which we name α_0. Then, the Chi model of Equation (**??**) is equivalent to the following model:

$$n = 0 \wedge V = 0 \gg \alpha_0 \parallel \alpha_1$$

In MULTIFORM deliverable D.1.2.1 [**?**] we formalize the translation of process translating process algebraic constructs to CIF.

CIF operational semantics is inspired by that of Chi 2.0 [**?**]. For instance, the idea of using guard trajectories for defining urgency was taking from the semantics of the latter formalism. However CIF and Chi differ in several aspects: for instance in CIF parallel composition is a restriction on the behavior of the components, synchronization is defined in a different way, and the notion of control variables is not present in Chi.

6.2. Indirect mapping of other formalisms to and from CIF

So far we have shown examples of models in different formalisms, which could be (easily) translated to CIF. Thus, given two formalisms, it is possible to identify

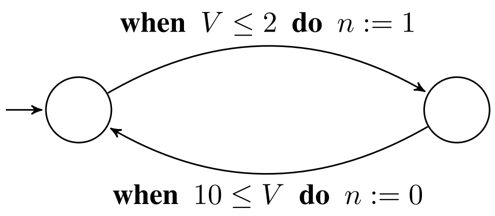


Figure 30: CIF representation of the discrete part of the tank Chi controller model (α_1).

subsets of these which can be translated to each other. Figure **??** illustrates this, where the big circles represent the set of models that can be expressed with a formalisms, and colored circles are the language subsets. In this figure, the two yellow circles correspond to the subset that can be translated, which is expressed by means of an arrow connecting them.

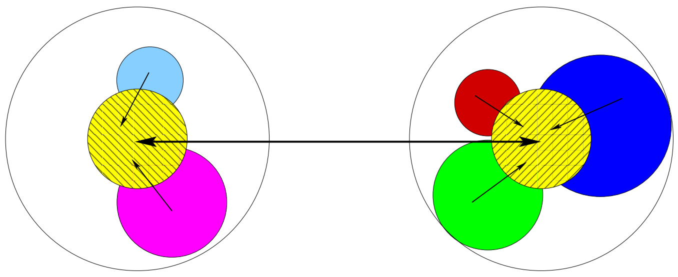


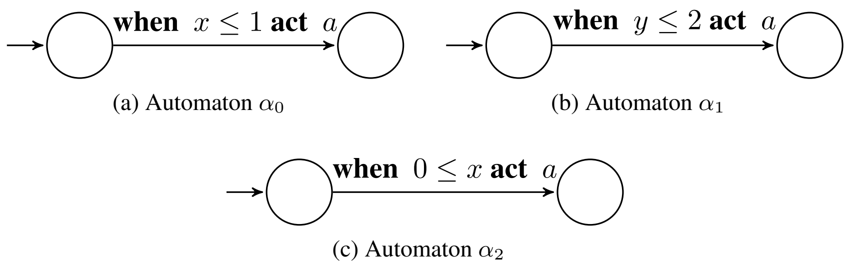
Figure 31: Model transformations between two formalisms.

However, there are elements of other formalisms that cannot be directly translated to CIF. Conversely, CIF contains operators such as urgency, synchronization, and control variables that are not natively supported by may formalisms to which CIF has to be translated to. To overcome this problem, it is possible to rewrite models using other constructs that are supported in the target language (see Figure **??**). In particular, certain operators can be eliminated in terms of other CIF constructs. For instance, urgency can be implemented by means of the tcp predicates. In the context of CIF this process is called process-algebraic linearization, and it was formalized in [**?**], and implemented in [**?**].

We illustrate what we have stated above with an example. Equation (**??**) shows a CIF composition, featuring the use of synchronization and urgency operators, where automata α_i, $i \in \{0, 1, 2\}$ are depicted if Figure **??**.

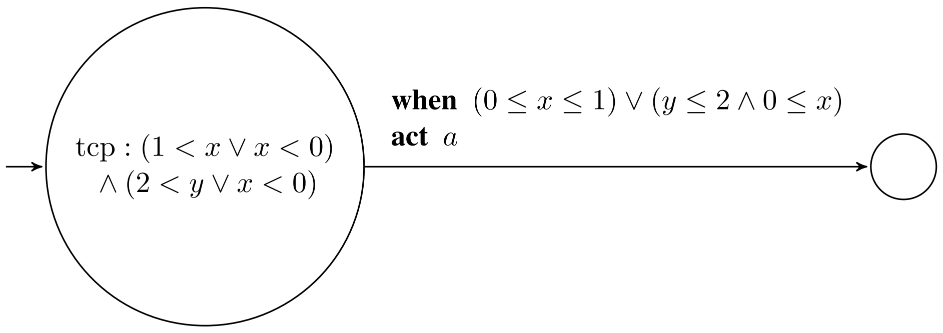
$$\upsilon_a(\gamma_a(\alpha_0 \parallel \alpha_1) \parallel \gamma_a(\alpha_2)) \tag{10}$$

Most formalisms do not support the possibility of specifying urgency and synchronization constraints. In such cases, the model of Equation (**??**) cannot be translated directly. However, the operators can be eliminated, resulting in the automaton shown in Figure **??**.



when $x \leq 1$ **act** a
(a) Automaton α_0
when $y \leq 2$ **act** a
(b) Automaton α_1
when $0 \leq x$ **act** a
(c) Automaton α_2

Figure 32: Three CIF automata that synchronize on urgent action a.



tcp : $(1 < x \vee x < 0)$ $\wedge\,(2 < y \vee x < 0)$
when $(0 \leq x \leq 1) \vee (y \leq 2 \wedge 0 \leq x)$
act a

Figure 33: Linearization of the three automata

In general, the state space may become large as a result of the linearization process. We show in [**?**] how this may be circumvented by means of variables (called location pointers) that keep track of the active locations of the intervening automata.

6.3. Limitations of the discussed formalisms as interchange format

In this section we discuss why CIF is necessary as an interchange format, instead of using one of the existing formalisms for this purpose.

Being a formalism for specifying timed systems, Uppaal does not provide support for concepts such as differential equations, or urgency conditions over differential variables. However, an interchange format for hybrid languages needs to have these features.

SpaceEx only provides support for a limited for of urgency, which can be specified by means of invariants. For instance, urgency conditions involving disjunctions, or global urgency conditions are not supported by SpaceEx. However languages such as Uppaal, gPROMS, or Chi, support more sophisticated forms of urgency. CIF can deal with these forms.

In addition, synchronization in SpaceEx is limited to connected actions. CIF provides more flexibility in this regard, as shown in Section **??**.

Discrete variables are not supported either, which makes it inconvenient for transformations to other formalisms that have this feature.

gPROMS does not support concepts such as action synchronization, control variables, and urgency across parallel composition and synchronizing actions. We have seen that target formalisms such as Uppaal or SpaceEx provide these features, and thus the importance of supporting them.

7. Validation of the semantics

The SOS rules presented in this work are basically definitions. As such, it is not possible to talk about mathematical correctness of the specification. However, there are important properties that we want to be valid in our semantics. The first of these properties is that bisimulation is a congruence for the operators. This is important because it enables us to replace equivalent compositions without changing the meaning of the whole model, making this property essential for reasoning about models [**?**]. Secondly, we prove properties on the transition system induced by the deduction rules given in Section **??**. These properties validate our insights about the three kinds of transitions. Lastly, we enunciate a number of properties that reflect our intuitive understanding of CIF operators, and we check that the SOS rules guarantee these properties.

7.1. Stateless bisimilarity

To compare two compositions, we use the notion of *stateless bisimilarity*, which is the most robust equivalence for transitions systems with data [**?**]. Next, we enunciate the definition of stateless bisimulation.

Definition 10 (Stateless bisimulation)**.** *A symmetric relation $R \subseteq \mathcal{C} \times \mathcal{C}$ is called a stateless bisimulation relation if and only if for all $(p, q) \in R$ the following transfer conditions hold.*

- $\langle \forall \sigma, a, b, X, p', \sigma' :: (p, \sigma) \xrightarrow{a,b,X} (p', \sigma') \Rightarrow \langle \exists q' :: (q, \sigma) \xrightarrow{a,b,X} (q', \sigma') \wedge (p', q') \in R \rangle \rangle$

- $\langle \forall \sigma, \rho, A, \theta, p', \sigma' :: (p, \sigma) \overset{\rho, A, \theta}{\longmapsto} (p', \sigma') \Rightarrow \langle \exists q' :: (q, \sigma) \overset{\rho, A, \theta}{\longmapsto} (q', \sigma') \wedge (p', q') \in R \rangle \rangle$

- $\langle \forall \sigma, A, p', \sigma' :: (p, \sigma) \overset{A}{\dashrightarrow} (p', \sigma') \Rightarrow \langle \exists q' :: (q, \sigma) \overset{A}{\dashrightarrow} (q', \sigma') \wedge (p', q') \in R \rangle \rangle$

Two CIF compositions p and q are said to be bisimilar, denoted as $p \underline{\leftrightarrow} q$, if and only if there exists a stateless bisimulation relation R such that $(p, q) \in R$.

An important property of the semantics is that it enables algebraic reasoning by allowing to replace equivalent (bisimilar) compositions. This is formalized in the following theorem.

Theorem 1 (Bisimulation is a congruence)**.** *Bisimulation is a congruence for all CIF operators.*

Proof. The above theorem can be proved by noticing that all CIF SOS rules are in *process-tyft* format, which guarantees the congruence for stateless bisimilarity [**?**]. □

7.2. Properties of the hybrid transition systems

When designing the SOS rules, one has in mind certain *invariants* for the different kind of transitions. These invariants are properties that must always hold for every transition, and they were informally described in Section **??**. For instance, for every environment transition $(p, \sigma) \overset{A}{\dashrightarrow} (p', \sigma')$, we want A to be the set of synchronizing actions of compositions p and p'. Similarly, every state of the transition system induced by the SOS rules should be a consistent one, according Definition **??** (Consistency). In this section, we give a formal statement of such properties. In **??** and **??** we prove that the SOS rules, defined in Section **??**, induce hybrid transition systems that satisfy these properties.

The first property states that environment transitions correctly capture the notion of consistency. Given an environment transition $(p, \sigma) \overset{A}{\dashrightarrow} (p', \sigma')$, we want valuation σ (σ') to be consistent with composition p (p' respectively). Using Definition **??**, we can state this as follows.

Property 1 (Consistency in environment transitions)**.** *For all p, p', σ, σ', and A we have:*

$$(p, \sigma) \overset{A}{\dashrightarrow} (p', \sigma') \Rightarrow \sigma \models p \wedge \sigma' \models p'$$

For the next property we need to define the set of synchronizing actions of a composition.

Definition 11 (Set of synchronizing actions)**.** *The set of synchronizing actions of a composition is defined recursively as follows.*

$sync((V, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, E, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype})) = \mathrm{act}_S$
$sync(p \parallel q) = sync(p) \cup sync(q)$
$sync(\gamma_a(p)) = \{a\} \cup sync(p)$
$sync(u \gg p) = sync(p)$
$sync(|[_{\mathrm{V}}\, x = e, \dot{x} = e :: p\,]|) = sync(p)$
$sync(|[_{\mathrm{A}}\, a :: p\,]|) = sync(p) \setminus \{a\}$
$sync(\upsilon_{a_\tau}(p)) = sync(p)$
$sync(\mathrm{D}_{x:G}(p)) = sync(p)$
$sync(\mathrm{ctrl}_x(p)) = sync(p)$

Using the definition above, we can express the property that states that the label of environment transitions coincides with function $sync(p)$ and $sync(p')$, which, in turn, implies that the set of synchronizing actions is not changed by an environment transition.

Property 2 (Synchronizing actions in environment transitions)**.** *For all p, p', σ, σ', and A we have:*

$$(p, \sigma) \overset{A}{\dashrightarrow} (p', \sigma') \Rightarrow sync(p) = A \wedge sync(p') = A$$

The environment transitions initialize the compositions by picking a unique active location for each automaton contained inside them. A property of environment transitions is that once this unique initial location is picked, it cannot be changed by another environment transition. The property is formulated below.

Property 3 (Closure of initialization)**.** *For all p, p', p'', σ, σ', ς, ς', A, and B we have:*

$$(p, \sigma) \overset{A}{\dashrightarrow} (p', \sigma') \wedge (p', \varsigma) \overset{B}{\dashrightarrow} (p'', \varsigma') \Rightarrow p' \equiv p''$$

The next property relates variable and guard trajectories, and valuations. We want the domain of the trajectories to be *closed intervals*, that the valuation in the initial state coincides with the initial valuation of variables at time 0, as given by the variable trajectory, and similarly, that the valuation in the final state is the same as the valuation of variables at the end time of the delay. This is formalized by the following property.

Property 4 (Trajectories and valuations)**.** *For all p, p', σ, σ', ρ, A, and θ we have:*

$$(p, \sigma) \overset{\rho, A, \theta}{\longmapsto} (p', \sigma') \Rightarrow \langle \exists t :: \mathrm{dom}(\rho) = [0, t] = \mathrm{dom}(\theta) \wedge \rho(0) = \sigma \wedge \rho(t) = \sigma' \rangle$$

Using the definitions of trajectory prefix and postfix, we can specify the properties of *prefix closure* and *postfix closure* for time transitions. Prefix closure states that if a time delay is possible, then it is possible to take a shorter time delay. Postfix closure states that the last part of a delay is also a valid delay in the system.

The properties of prefix and postfix closure give an intuitive meaning to time delays. A consequence of these properties is that a time delay can be split in as many parts as desired, yielding the same behavior as the one obtained if the whole delay was made. These properties are formalized below.

Property 5 (Prefix closure)**.** *For all p, p', σ, σ', ρ, A, θ, and for all time points $s, t \in \mathbb{T}$, such that $\mathrm{dom}(\rho) = [0, t]$, $0 < s$, and $s \leq t$, we have that for some σ'':*

$$(p, \sigma) \overset{\rho,A,\theta}{\longmapsto} (p', \sigma') \Rightarrow (p, \sigma) \overset{\rho^{\leq s},A,\theta^{\leq s}}{\longmapsto} (p', \sigma'')$$

Property 6 (Postfix closure)**.** *For all p, p', σ, σ', ρ, A, θ, and for all time points $s, t \in \mathbb{T}$, such that $\mathrm{dom}(\rho) = [0, t]$ and $s \leq t$, we have that for some σ'':*

$$(p, \sigma) \overset{\rho,A,\theta}{\longmapsto} (p', \sigma') \Rightarrow (p', \sigma'') \overset{\rho^{\geq s},A,\theta^{\geq s}}{\longmapsto} (p', \sigma')$$

Another property we are interested in is the *property of state* [**?**]. This property models the idea that the state of system contains all the relevant information about the past of a system, and thus future flows only depend on the current valuation. Using the definition of flow concatenation, it is possible to express the property of state.

Property 7 (State)**.** *For all p, p', p'', ρ, ρ', A, θ, θ', such that $\mathrm{dom}(\rho) = [0, t]$ and $\mathrm{dom}(\rho') = [0, t']$ and $\rho(t) = \rho'(0)$ we have that:*

$$(p, \rho(0)) \overset{\rho,A,\theta}{\longmapsto} (p', \rho(t)) \wedge (p', \rho'(0)) \overset{\rho',A,\theta'}{\longmapsto} (p'', \rho'(t')) \Rightarrow$$
$$(p, \rho(0)) \overset{(\rho \cdot_t \rho'),A,\theta \cdot_t \theta'}{\longmapsto} (p'', \rho'(t'))$$

7.3. Properties of the operators

This section presents properties of the CIF operators, which provide additional validation of the semantics. These properties not only enable algebraic reasoning, but are also important for rewriting CIF models to normal forms that are more suitable for a specific purpose. For instance, the commutativity of variable scope (Property **??**) allow us to push all the scoping operators to the outermost level, and this simplifies the linearization of CIF models, or its symbolic treatment. Note that it is not our intention to provide a sound and complete axiomatization of all the properties of CIF models.

We want the parallel composition to be commutative, and associative, since these are well know properties from literature [**?**], and essential for algebraic reasoning.

Property 8 (Properties of parallel composition)**.** *For all p, q, and r the following equivalences hold:*

$$\begin{array}{rcl} p \parallel q & \leftrightarrow & q \parallel p \\ (p \parallel q) \parallel r & \leftrightarrow & p \parallel (q \parallel r) \end{array}$$

In general, synchronization, urgency, control variable, and scope operators do not distribute over parallel composition. For the remaining operators we have the distributivity laws enunciated in the next property.

Property 9 (Distributivity of operators wrt. parallel composition)**.** *For all u, x, p, q, and G, the following equivalences hold:*

$$\begin{array}{rcl} \mathrm{D}_{x:G}(p) \parallel q & \leftrightarrow & \mathrm{D}_{x:G}(p \parallel q) \\ (u \gg p) \parallel q & \leftrightarrow & u \gg (p \parallel q) \end{array}$$

Some CIF operators can be exchanged in a model without altering its behavior. The commutativity properties of the CIF operators are summarized in Table **??**. A 0 in the position (f, g) indicates that the two operators are not commutative, whereas a 1 indicates that the operators are, i.e.:

$$f(g(p)) \leftrightarrow g(f(p))$$

for all compositions p. For instance, declaring first a as synchronizing, and then b, in a composition p, delivers a bisimilar composition to the one where first b is declared as synchronizing, and then a.

Property 10 (Commutativity among CIF operators)**.** *For all a, a', u, u', x, and x', the commutativity laws shown in Table* **??** *hold for any composition p.*

	$\gamma_a(\_)$	$u \gg \_$	$\|[_\mathrm{V}\, x :: \_\,]\|$	$\|[_\mathrm{A}\, a :: \_\,]\|$	$\upsilon_a(\_)$	$\mathrm{D}_x(\_)$	$\mathrm{ctrl}_x(\_)$
$\gamma_{a'}(\_)$	1	1	1	0	1	1	1
$u' \gg \_$	1	1	0	1	1	1	1
$\|[_\mathrm{V}\, x' :: \_\,]\|$	1	0	0	1	1	0	0
$\|[_\mathrm{A}\, a' :: \_\,]\|$	0	1	1	1	0	1	1
$\upsilon_{a'}(\_)$	1	1	1	0	1	1	1
$\mathrm{D}_{x'}(\_)$	1	1	0	1	1	1	1
$\mathrm{ctrl}_{x'}(\_)$	1	1	0	1	1	1	1

Table 2: Commutativity among different CIF operators.

All CIF operators but parallel composition and variable scope have the *closure property*, which states that applying an operator f twice to a composition p has the same effect as applying it once.

Property 11 (Closure of CIF operators)**.** *For all a, u, x, G, and p the following bisimilarity holds:*

$$f(f(p)) \leftrightarroweq f(p)$$

where $f \in \{\gamma_a(\_), u \gg \_, [\![_{\mathrm{A}}\, a :: \_\,]\!], \upsilon_a(\_), \mathrm{D}_{x:G}(\_), \mathrm{ctrl}_x(\_)\}$.

Note that closure of variable scope operator does not hold in general. The validity of the equivalence

$$[\![_{\mathrm{V}}\, x = e_0, \dot{x} = e_1 :: [\![_{\mathrm{V}}\, x = e_0, \dot{x} = e_1 :: p\,]\!]]\!] \leftrightarroweq [\![_{\mathrm{V}}\, x = e_0, \dot{x} = e_1 :: p\,]\!]$$

is guaranteed only if x and $\dot{x}$ do not occur free in expressions e_0 and e_1.

An important property of the initialization operator is that all the predicates are taken into account at the same time when initializing a CIF composition. In other words, we want the initialization operator to restrict only the initial state of a given system. If this is the case, then all the conjunction of all initialization predicates can be pushed to the outermost level. The following property, together with Properties **??** and **??** are crucial for achieving this.

Property 12 (Conjunction of initializations)**.** *For all u, v, and p the following equivalence holds:*

$$u \gg (v \gg p) \quad \leftrightarroweq \quad (u \wedge v) \gg p$$

The dynamic type operator has some interesting properties. The first one of them states that the dynamic type constraints in a composition are cumulative. This is, if in a composition the dynamic behavior of certain variable x is constrained by dynamic types G_0 and G_1, then variable x is constrained by the intersection of these two types. This, together with Property **??**, allows us to infer that dynamic type constraints in one component of the parallel composition also affect the other component behavior. This is formalized below.

Property 13 (Properties of dynamic types)**.** *For all x, G_0, G_1, and p the following equivalence holds:*

$$\mathrm{D}_{x:G_0}(\mathrm{D}_{x:G_1}(p)) \quad \leftrightarroweq \quad \mathrm{D}_{x:G_0 \cap G_1}(p)$$

The properties of the dynamic type operator ensure that, as with the initialization operator, all dynamic types in a composition are considered simultaneously in a time delay. In other words, dynamic types can be pushed to the outermost level of a parallel composition using intersection. Another consequence the property stated above is that if a variable is declared both as continuous and discrete in a certain composition, then the most restrictive dynamic type (discrete) will be taken into account.

The variable scope operator has several distributivity properties with regard to the different CIF operators. Together, they state that the variable scope operators of a composition can be pushed to the outermost level by using renaming, which captures the intuition behind variable declarations. Given an expression e, and variables x_i, y_i, $0 \leq i \leq N$, for some positive constant N, expression $e[y_0, \ldots, y_{n-1}/x_0, \ldots x_{n-1}]$ is the result of replacing all occurrences of variable x_i by y_i in expression e.

Property 14 (Commutativity of variable scope)**.** *Let y and $\dot{y}$ be variables that do not appear in the set of free variables of expressions e_0 and e_1, compositions p, and q, and predicate u, such that $y \neq z$, then the following equivalences hold:*

$$\begin{aligned}
&|[_{\mathrm{V}}\, x = e_0, \dot{x} = e_1 :: p\ ]| \parallel q \;\underline{\leftrightarrow} \\
&\qquad\qquad |[_{\mathrm{V}}\, (x = e_0, \dot{x} = e_1)[y, \dot{y}/x, \dot{x}] :: p[y, \dot{y}/x, \dot{x}] \parallel q\ ]| \\
&u \gg |[_{\mathrm{V}}\, x = e_0, \dot{x} = e_1 :: p\ ]| \;\underline{\leftrightarrow} \\
&\qquad\qquad |[_{\mathrm{V}}\, (x = e_0, \dot{x} = e_1)[y, \dot{y}/x, \dot{x}] :: u \gg p[y, \dot{y}/x, \dot{x}]\ ]| \\
&\mathrm{D}_{z:G}(|[_{\mathrm{V}}\, x = e_0, \dot{x} = e_1 :: p\ ]|) \;\underline{\leftrightarrow} \\
&\qquad\qquad |[_{\mathrm{V}}\, (x = e_0, \dot{x} = e_1)[y, \dot{y}/x, \dot{x}] :: \mathrm{D}_{z:G}(p[y, \dot{y}/x, \dot{x}])\ ]| \\
&\mathrm{ctrl}_z(|[_{\mathrm{V}}\, x = e_0, \dot{x} = e_1 :: p\ ]|) \;\underline{\leftrightarrow} \\
&\qquad\qquad |[_{\mathrm{V}}\, (x = e_0, \dot{x} = e_1)[y, \dot{y}/x, \dot{x}] :: \mathrm{ctrl}_z(p[y, \dot{y}/x, \dot{x}])\ ]|
\end{aligned}$$

A similar property holds for action scopes.

Property 15 (Commutativity of action scope)**.** *For all compositions p and q, and for all variables a, b, and c, such that c does not appear in the set of free variables of compositions p and q, and $a \neq c$, the following bisimilarities hold:*

$$\begin{aligned}
&|[_{\mathrm{A}}\, b :: p\ ]| \parallel q \;\underline{\leftrightarrow}\; |[_{\mathrm{A}}\, c :: p[c/b] \parallel q\ ]| \\
&\upsilon_a(|[_{\mathrm{A}}\, b :: p\ ]|) \;\underline{\leftrightarrow}\; |[_{\mathrm{A}}\, c :: \upsilon_a(p[c/b])\ ]| \\
&\gamma_a(|[_{\mathrm{A}}\, b :: p\ ]|) \;\underline{\leftrightarrow}\; |[_{\mathrm{A}}\, c :: \gamma_a(p[c/b])\ ]|
\end{aligned}$$

8. Examples

We illustrate CIF by means of two examples. In Section **??**, we present a model of unmanned underwater and aerial vehicles, to show how different forms of communication can be achieved in CIF. In Section **??**, the use of urgency is explained using a simple producer-consumer model.

In the remainder of this section we use the following conventions:

- Graphically, the sets of synchronizing actions, control variables, and dynamic type mapping of an automaton are declared in a rectangle, above the top-right corner of the rectangle that encloses the picture of the automaton (see Figure **??**). The set of control variables is specified using the $\mathbf{control}$ keyword. The set of synchronizing actions is specified using the $\mathbf{sync}$ keyword. The dynamic type of a variable is specified using three different keywords: $\mathbf{disc}$, $\mathbf{clock}$, and $\mathbf{cont}$. In the examples that follow, we associate a dynamic type to the model variables only at the top level composition (for instance Figure **??**). If there is a dynamic type declaration in the automaton we assume the variable to be local. For local variables, the dynamic type and the control variable declarations can be combined. According to the conventions previously mentioned, for the automaton of Figure **??** we have: $\mathrm{var}_C = \{x, c, z\}$, $\mathrm{act}_S = \{a, b\}$, $\mathrm{dtype} = \{(y, G_{disc}), (c, G_{clock}), (z, G_{cont})\}$. Moreover, we assume automaton $AutName$ to be enclosed by variable scope operators that declare variables y, c, and z as local.

- To keep the models concise, we omit the set of jumping variables, and we assume it to be all the step variables controlled by the automaton that appear in the update predicate. For instance, assume only variables x and z are controlled in automaton α. Then we write

  $$\textbf{when } g \textbf{ act } a \textbf{ do } x^+ = y^+ + y + z^+$$

  instead of

  $$\textbf{when } g \textbf{ act } a \textbf{ do } (\{x, z\}, x^+ = y^+ + y + z^+).$$

- When passing data using synchronizing actions and shared variables, we use channel notation to distinguish the sender ($h\ !\ e$) and the receiver ($h\ ?\ x$). Thus we make use of the following definitions:

  $$\begin{array}{lcl} \textbf{act } h\ !\ e \textbf{ do } r & \triangleq & \textbf{act } h \textbf{ do } v_h^+ = e \wedge r \\ \textbf{act } h\ ?\ x \textbf{ do } r & \triangleq & \textbf{act } h \textbf{ do } v_h^+ = x^+ \wedge r \end{array}$$

  where we assume a variable v_h is declared for each action h.

8.1. Unmanned underwater and aerial vehicles

In this example, we want to model the communication between aerial vehicles (*AV*), underwater vehicles (*UV*), and a command center (*CC*). The command center generates commands that are to be transmitted to the underwater vehicles by the

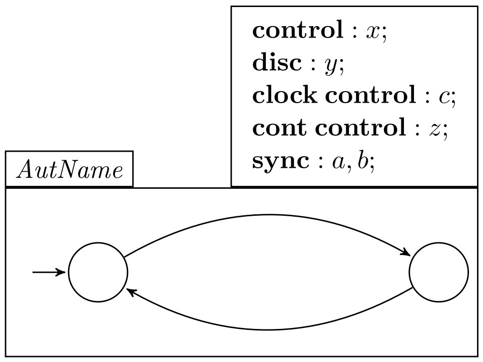


Figure 34: Automaton with declarations.

aerial vehicle. Every underwater vehicle waits for a command to carry out. Once it receives it, it submerges and executes the task.

Each command is addressed to a specific underwater vehicle, which is identified by a unique id. We assume the aerial vehicle flies along a straight line, looking for the corresponding underwater vehicle. The command center is located at position 0, and N is the maximum distance the AV can be away from the control center. Communication between the aerial vehicle and an underwater vehicle is successful if they are at most R units of distance away from each other, and the underwater vehicle is on the water surface.

Figure **??** shows the model of the command center. Initially it is in the $Ready$ state, and sends an order to the aerial vehicle, which in turn will pass it to a specific underwater vehicle. After sending, the command center waits for a reply from the aerial vehicle that conveys information about the success of the command. In this model, an order is abstracted as the recipient's id (0, 1, or 2).

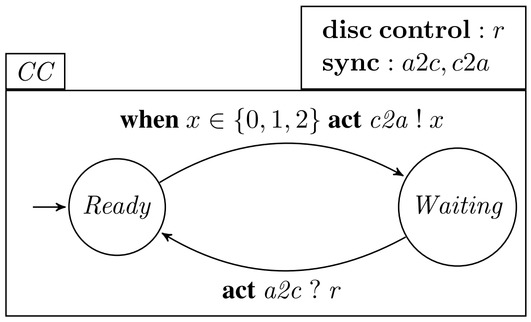


Figure 35: Command center.

The model of the aerial vehicle is depicted in Figure **??**. The aerial vehicle is initially at position 0, waiting for an order from the command center. After receiving the order, the vehicle starts searching for an underwater vehicle to pass the order to. Variable x_{av} represents the vehicle position along the surface of the water. In the *Searching* position the rate of change of x_{av} is determined by variable v. Since x_{av} is controlled by the AV, no other process can change its value. Variable x_{prev} records the last position where the AV tried to establish a communication. Every Δ units of distance, a message is broadcast using the action $a2v$, after which the AV waits T_w time units for an acknowledge message. If an acknowledge is received the AV returns to position 0 and notifies the control center of the successful delivery. Otherwise the AV continues flying Δ units away from the last point it tried to send the command, and the procedure is repeated again. If the vehicle has flown beyond the search limits ($N \leq x_{av}$), then it starts flying back to position 0 while still searching for the corresponding UV.

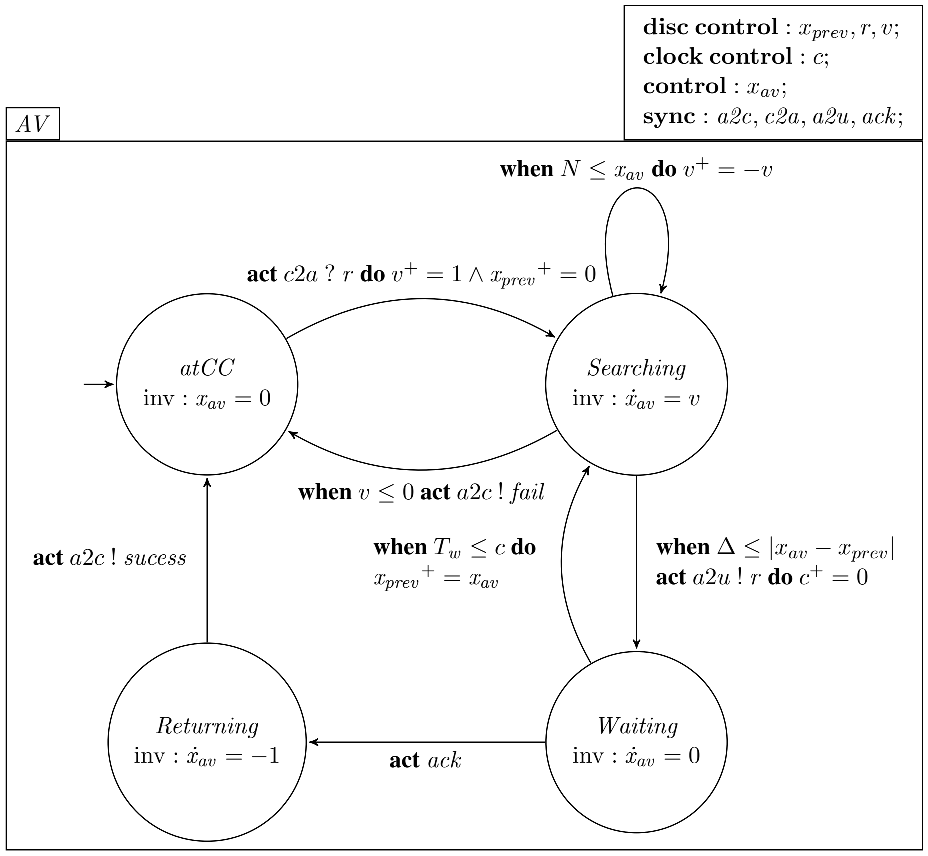

disc control : x_{prev}, r, v;
clock control : c;
control : x_{av};
sync : $a2c$, $c2a$, $a2u$, ack;
AV
when $N \leq x_{av}$ **do** $v^+ = -v$
act $c2a$? r **do** $v^+ = 1 \wedge x_{prev}{}^+ = 0$
$atCC$
inv : $x_{av} = 0$
$Searching$
inv : $\dot{x}_{av} = v$
when $v \leq 0$ **act** $a2c$! $fail$
act $a2c$! $sucess$
when $T_w \leq c$ **do** $x_{prev}{}^+ = x_{av}$
when $\Delta \leq |x_{av} - x_{prev}|$ **act** $a2u$! r **do** $c^+ = 0$
$Returning$
inv : $\dot{x}_{av} = -1$
act ack
$Waiting$
inv : $\dot{x}_{av} = 0$

Figure 36: Aerial vehicle.

Figure **??** models the behavior of the underwater vehicles. Initially, each vehicle is at some random position on the water surface. It waits for an order of the AV. If the order is addressed to the recipient of the message, then it sends an acknowledge message, and it submerges to carry out the task. Otherwise the vehicle does not reply. Each command takes at least M time units to be executed. After that time has elapsed the vehicle *can* surface, or it can continue submerged at most G time units more (then it *must* surface). These conditions are enforced by putting a guard on action $surface$, and a tcp predicate in location $Submerged$. Upon completion of a command, the UV emerges at an unspecified position. The guard of the self-loop at location $OnSurface$ represents the situation in which a message from the AV is not received, since the distance between sender and receiver is too long. Similarly, the self-loop at the $Submerged$ location models the fact that the message from the AV is dropped. These self-loops are necessary to prevent the AV from blocking on a broadcast message using action $a2u$.

The models presented above are composed as shown in Figure **??**. Here two different forms of communications are used. Actions $a2c$, $c2a$, and ack are used to implement point to point channels. Action $a2u$, on the other hand, is used to implement broadcast communication from the aerial vehicle to the underwater vehicles.

8.2. Producer-consumer

The illustration presented in this section is intended to show how the urgency operator can be used to obtain both patient and impatient synchronization [**?**]. The difference between these forms of synchronization can be understood only in the context of parallel composition. Informally, a composition p that wants to synchronize in a patient action a, waits for the other components to be ready to execute this action, and in this way, p allows time delays. On the other hand, a component that wants to synchronize on an impatient action a does not wait for the other partners to be ready, and disables the passage of time. We give an example of this next.

Consider a producer that sends products to a buffer, and a consumer that retrieves the contents from the buffer. The processes operate at different rates, and situations can arise where the producer cannot place its product in the buffer because it is full, or the consumer finds the buffer empty. In this example we analyze the first situation.

There are cases where a producer not being able to send a product is not critical, for instance a video streaming protocol in which frames can be dropped. In this case, we may be interested in the average amount of dropped packages. If we are to model such systems, then patient synchronization is useful.

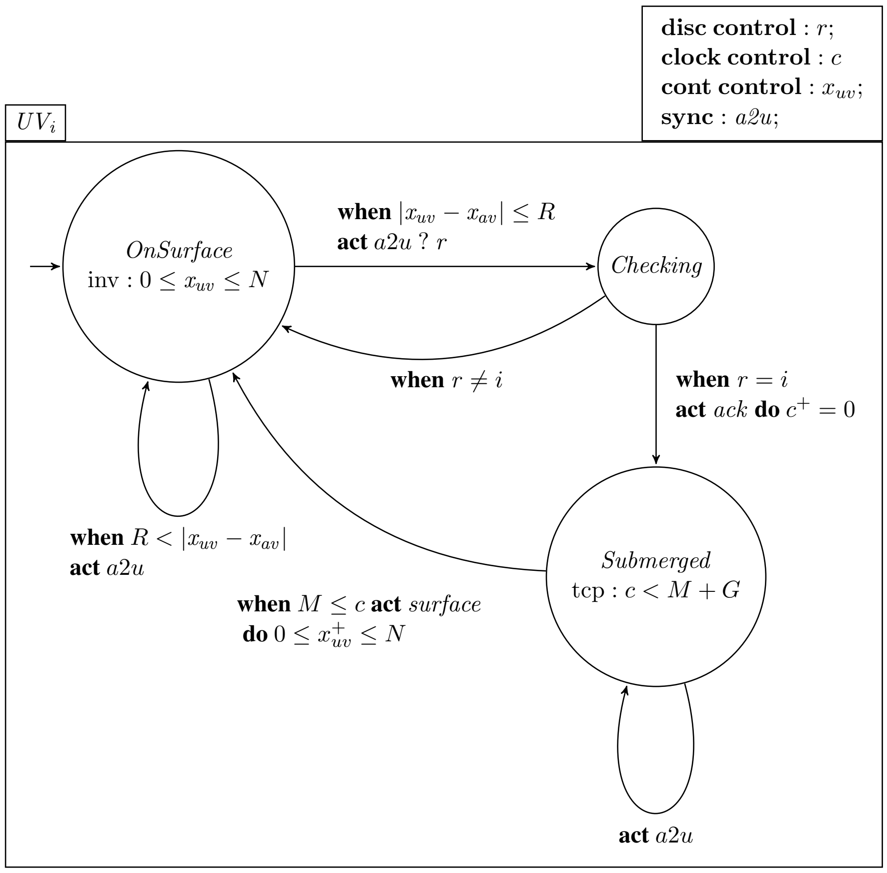

disc control : r;
clock control : c
cont control : x_{uv};
sync : $a2u$;
UV_i
$OnSurface$
inv : $0 \leq x_{uv} \leq N$
when $|x_{uv} - x_{av}| \leq R$
act $a2u \mathbin{?} r$
$Checking$
when $r \neq i$
when $r = i$
act ack **do** $c^+ = 0$
when $R < |x_{uv} - x_{av}|$
act $a2u$
when $M \leq c$ **act** $surface$
do $0 \leq x_{uv}^+ \leq N$
$Submerged$
tcp : $c < M + G$
act $a2u$

Figure 37: Underwater vehicle.

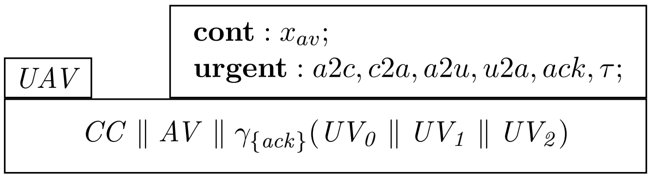

cont : x_{av};
urgent : $a2c, c2a, a2u, u2a, ack, \tau$;
UAV
$CC \parallel AV \parallel \gamma_{\{ack\}}(UV_0 \parallel UV_1 \parallel UV_2)$

Figure 38: Top level model: unmanned underwater and aerial vehicles.

In other cases, the scenario where the producer is not able send its product should not occur. For instance, a delayed signal in a car control system may cause a serious failure in the system. In this case, it is possible to model this requirement by means of impatient synchronization.

First, we make an high-level model of a simple producer-consumer protocol, where it is possible for a producer to miss its deadline. Next we turn our attention to the case where such a failure cannot be tolerated.

Figure **??** shows the model of the producer. It requires T time units to have a product ready. The model of the buffer is shown in Figure **??**. The buffer has a

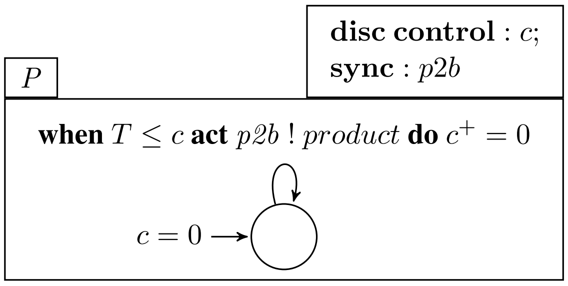
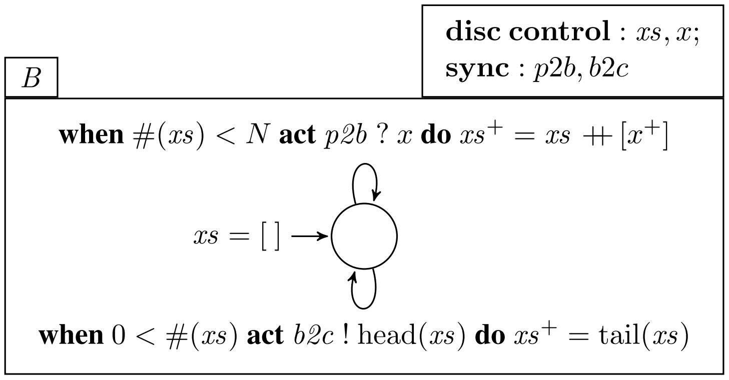


Figure 39: Model of the producer.

capacity N. It can receive a product if it is not at the limit of its storing capacity, and similarly it can send a product if it is not empty. Received products are stored in the list xs. Given a list xs, $\#(xs)$ returns the length of xs, $\mathrm{head}(xs)$ returns the first element of xs, $\mathrm{tail}(xs)$ is the list that results from removing the first element from xs, and $xs \mathbin{+\!\!+} ys$ is the list that is obtained after appending ys to the end of xs.



B
disc control : xs, x;
sync : $p2b, b2c$
when $\#(xs) < N$ act $p2b$? x do $xs^+ = xs \mathbin{+\!\!+} [x^+]$
$xs = [\,]$
when $0 < \#(xs)$ act $b2c$! $\mathrm{head}(xs)$ do $xs^+ = \mathrm{tail}(xs)$

Figure 40: Model of the buffer with capacity N.

If an undelivered product is not considered a critical failure, then we can compose the models of the producer, consumer, and buffer as shown in Figure **??**, where the model of the consumer is omitted since it is not relevant for our discussion. Here, since action $p2b$ is urgent in the parallel composition, this means that time can progress up to the point where condition $T \leq c \wedge \#(xs) < N$ holds. We could have also used the tcp predicate to achieve the same behavior, however for the urgent action $p2b$, the use of the operator is more convenient since otherwise it is necessary to modify the tcp predicates of the producer and consumer, in a non-compositional way.

$PCpatient$

$$\boxed{\upsilon_{p2b}(P \parallel B \parallel C)}$$

Figure 41: Producer-consumer: patient synchronization.

If we consider a model where the producer always has to meet its deadline, then we can use impatient synchronization. This can be modeled in CIF as shown in Figure **??**. Here, the urgency condition becomes $T \leq c$, and if synchronizing action $p2b$ cannot take place by the time it is satisfied, then no further time delays are possible (which could lead to a deadlock if the buffer is full and the consumer cannot take a product out of the buffer without any time delays).

$PCimpatient$

$$\boxed{\upsilon_{p2b}(P) \parallel B \parallel C}$$

Figure 42: Producer-consumer: impatient synchronization.

9. Conclusions

We have presented the formal definition of a compositional interchange format for hybrid systems. This definition is given in terms of SOS rules, which allow us to prove compositionality of the operators of the language. The semantics is given in such a way that it is independent of implementation concepts, and it is validated by proving axiomatic properties.

CIF features operators which define concepts in an orthogonal way. These operators can be combined to model various aspects of discrete-event, timed and hybrid systems. For instance, the example of the unmanned vehicles shows how different forms of communication can be achieved in CIF using a single operator. Similarly, the producer-consumer example shows how the combination of the urgency operator and the synchronization operator can be used to implement patient and impatient synchronization.

The use of structured operational semantics in the definition of CIF plays an essential role in the proofs of compositionality, and at the same time it gives a straightforward semantics to the language. We have derived a simulator from this SOS specification, which shows the usefulness of the framework for practical applications [**?**].

In recent work [**?**], CIF was extended with hierarchy, which adds more flexibility to the language. The preservation of the properties presented in this work still needs to be shown for hierarchical CIF.

Based on the formal specification of the language, several semantic preserving transformations to and from CIF [**? ?**] have already been defined. As future work, we plan to define additional semantic preserving transformations between CIF and other formalisms, in particular, between CIF and hybrid automata.

The implementation of different forms of synchronization by means of a parallel composition operator is dealt with in [**?**]. In that work, the authors distinguish between synchronous (active) and asynchronous (passive) actions. Such a distinction is useful in the context of supervisory control theory. Furthermore, the authors consider an operator for restricting the scope of the interaction, using an action scope operator similar to the one presented in Section **??**; and they consider the possibility of multi-way synchronization. Unlike the parallel composition operator in [**?**], we do not embed synchronization information in it. Synchronization is achieved by means of the synchronizing action operator. As a result, we do not need conditions to guarantee commutativity, and associativity of the parallel composition operator, as shown in Section **??**. Point to point or multi-way synchronization can be obtained by changing the application order of the operator, instead of using a set of actions in the parallel composition operator. We believe this yields a simpler semantics, without shifting the complexity of synchronization considerations to the parallel composition operator. However, our synchronization operator is more limited since all the actions are active (synchronous), or they do not synchronize at all. We acknowledge the importance of the distinction between active and passive actions. The theoretical extension of CIF for its use in the context of supervisory control is something that we regard as future work.

Acknowledgments

The authors would like to thank Jos Baeten, Dennis Hendriks, and Michel Reniers for reading the draft version of this paper thoroughly, and providing detailed feedback. They also thank Dennis Hendriks for implementing the CIF2 simulator. In addition, the authors extend their gratitude to Pieter Collins, Albert Hofkamp, Mohammad Reza Mousavi for their valuable advice while developing this work; and to Ramon Schiffelers and the group of Sebastian Engell, in particular Christian Sonntag, for their valuable contributions to CIF in the HYCON, HYCON2, and Multiform projects.

References

[] G. Frehse, PHAVer: Algorithmic verification of hybrid systems past HyTech, in: M. Morari, L. Thiele (Eds.), Hybrid Systems: Computation and Control, 8th International Workshop, volume 3414 of *Lecture Notes in Computer Science*, Springer-Verlag, 2005, pp. 258–273.

[] K. G. Larsen, P. Pettersson, W. Yi, UPPAAL in a Nutshell, International Journal on Software Tools for Technology Transfer 1 (1997) 134–152.

[] R. R. H. Schiffelers, D. A. v. Beek, K. L. Man, M. A. Reniers, J. E. Rooda, A hybrid language for modeling, simulation and verification, in: S. Engell, H. Guéguen, J. Zaytoon (Eds.), IFAC Conference on Analysis and Design of Hybrid Systems, Saint-Malo, Brittany, France, pp. 235–240.

[] D. A. v. Beek, M. A. Reniers, R. R. H. Schiffelers, J. E. Rooda, Foundations of a compositional interchange format for hybrid systems, SE Report 2006-05, Eindhoven University of Technology, Systems Engineering Group, Department of Mechanical Engineering, Eindhoven, The Netherlands, 2006.

[] D. A. v. Beek, M. A. Reniers, J. E. Rooda, R. R. H. Schiffelers, HYCON Handbook of Hybrid Systems Control: Theory - Tools - Applications, Elsevier.

[] H. Beohar, D. E. Nadales Agut, D. A. van Beek, P. J. L. Cuijpers, Hierarchical states in the Compositional Interchange Format, in: L. Aceto, P. Sobocinski (Eds.), SOS, volume 32 of *Electronic Proceedings in Theoretical Computer Science*, 2010, pp. 42–56.

[] G. D. Plotkin, A structural approach to operational semantics, Journal of Logic and Algebraic Programming 60-61 (2004) 17–139.

[] M. R. Mousavi, M. A. Reniers, J. F. Groote, Notions of bisimulation and congruence formats for SOS with data, Information and Computation 200 (2005) 107–147.

[] MoBIES team, HSIF Semantics, Technical Report, University of Pennsylvania, 2002. Internal document.

[] A. Pinto, L. P. Carloni, R. Passerone, A. L. Sangiovanni-Vincentelli, Interchange format for hybrid systems: Abstract semantics, in: J. P. Hespanha, A. Tiwari (Eds.), Hybrid Systems: Computation and Control, 9th International Workshop, volume 3927 of *Lecture Notes in Computer Science*, Springer-Verlag, Santa Barbara, 2006, pp. 491–506.

[] Columbus IST project, http://www.columbus.gr, 2006.

[] S. D. Cairano, A. Bemporad, M. Kvasnica, An Architecture for Data Interchange of Switched Linear Systems, Technical Report D 3.3.1, HYCON NoE, 2006.

[] HYCON Network of Excellence, Hybrid control: Taming heterogeneity and complexity of networked embedded systems, `http://www.ist-hycon.org/`, 2005.

[] D. A. v. Beek, M. A. Reniers, R. R. H. Schiffelers, J. E. Rooda, Foundations of an interchange format for hybrid systems, in: A. Bemporad, A. Bicchi, G. Butazzo (Eds.), Hybrid Systems: Computation and Control, 10th International Workshop, volume 4416 of *Lecture Notes in Computer Science*, Springer-Verlag, Pisa, 2007, pp. 587–600.

[] W. P. M. H. Heemels, B. d. Schutter, A. Bemporad, Equivalence of hybrid dynamical models, Automatica 37 (2001) 1085–1091.

[] S. Di Cairano, A. Bemporad, An equivalence result between linear hybrid automata and piecewise affine systems, in: Proc. 45th IEEE Conference on Decision and Control, San Diego, pp. 2631 – 2636.

[] HYCON 2 consortium, Highly-complex and networked control systems, `http://www.hycon2.eu/`, 2010.

[] ITEA Twins consortium, Optimizing HW-SW co-design flow for sofware intensive system development, `http://www.twins-itea.org/`, 2009.

[] Embedded Systems Institute, Darwin project, `http://www.esi.nl/site/projects/darwin.html`, 2006.

[] C4C consortium, Control for coordination of distributed systems, `http://www.c4c-project.eu/`, 2008.

[] MULTIFORM consortium, Integrated multi-formalism tool support for the design of networked embedded control systems MULTIFORM, `http://www.multiform.bci.tu-dortmund.de`, 2008.

[] C. Sonntag, M. Hüfner, On the connection of equation- and automata-based languages: Transforming the Compositional Interchange Format to Modelica, in: Proc. 18th IFAC World Congress, pp. 12515–12520.

[] M. Hüfner, C. Sonntag, A. Jabrayilov, Finalization of the model exchange with Modelica and gPROMS, Technical Report, MULTIFORM consortium, 2010.

[] M. Hüfner, C. Sonntag, Deliverable D1.5.1 Realization of an interface to MUSCOD-II, Technical Report, MULTIFORM consortium, 2011.

[] D. E. Nadales Agut, M. A. Reniers, R. R. H. Schiffelers, K. Y. Jørgensen, D. A. v. Beek, A semantic-preserving transformation from the Compositional Interchange Format to UPPAAL, in: 18th Triennial World Congress of the International Federation of Automatic Control, Milano. CD-ROM.

[] D. A. van Beek, A. David, D. Hendriks, K. Y. Jörgensen, D. E. Nadales Agut, Deliverable D1.3.2 Software for translations between the CIF and UPPAAL, Technical Report, MULTIFORM consortium, 2011.

[] M. Goyal, G. Frehse, Translation between CIF and SpaceEx/PHAVer, Technical Report, MULTIFORM consortium, 2011.

[] C. Sonntag, Deliverable D1.4.2 Realisation of the Model Exchange of a Restricted Class of Models to Matlab/Simulink, Technical Report, MULTIFORM consortium, 2011.

[] The MathWorks, Inc, Writing S-functions, version 6, `http://www.mathworks.com`, 2005.

[] C. Sonntag, S. Fisher, Translating sequential function charts to the Compositional Interchange Format for hybrid systems, in: Proc. 49th IEEE Conference on Decision and Control, pp. 4250–4256.

[] K. H. John, M. Tiegelkamp, IEC 61131-3: Programming Industrial Automation Systems, Springer-Verlag, second edition, 2010.

[] J. Markovski, K. G. M. Jacobs, D. A. van Beek, L. J. A. M. Somers, J. E. Rooda, Coordination of resources using generalized state-based requirements, in: 10th International Workshop on Discrete Event Systems, pp. 300–305.

[] R. Su, D. A. van Beek, J. E. Rooda, Deliverable D2.2.1 Report on methods and prototype tool for the supervisory control of complex systems, Technical Report, MULTIFORM consortium, 2010.

[] Systems Engineering Group TU/e, Overview of supervisory control tools, `http://se.wtb.tue.nl/sewiki/supcon/cif_svc_overview`, 2011.

[] R. J. M. Theunissen, R. R. H. Schiffelers, D. A. v. Beek, J. E. Rooda, Supervisory control synthesis for a patient support system, in: Proceedings of the European control conference, Budapest, Hungary, pp. 4647–4652.

[] S. Fisher, D. A. van Beek, R. J. M. Theunissen, D. Hendriks, C. Sonntag, Deliverable D2.4.1 Report on the test results of the synthesis tool chain, Technical Report, MULTIFORM consortium, 2011.

[] EtherCAT Technology Group, EtherCAT - internet for Control Automation Technology, `http://www.ethercat.org/`, 2011.

[] D. A. v. Beek, P. Collins, D. E. Nadales, J. E. Rooda, R. R. H. Schiffelers, New concepts in the abstract format of the Compositional Interchange Format, in: A. Giua, C. Mahuela, M. Silva, J. Zaytoon (Eds.), 3rd IFAC Conference on Analysis and Design of Hybrid Systems, Zaragoza, Spain, pp. 250–255.

[] J. C. M. Baeten, D. A. van Beek, D. Hendriks, A. T. Hofkamp, D. E. Nadales Agut, J. E. Rooda, R. R. H. Schiffelers, Definition of the Compositional Interchange Format, Technical Report Deliverable D1.1.2, MULTIFORM consortium, 2010.

[] Systems Engineering Group TU/e, CIF 2 tooling, `http://devel.se.wtb.tue.nl/trac/cif`, 2011.

[] Object Management Group, Query/View/Transformation (QVT), version 1.0, `http://www.omg.org/spec/QVT/1.0/`, 2009.

[] D. Hendriks, R. R. H. Schiffelers, M. Hüfner, C. Sonntag, A transformation framework for the Compositional Interchange Format for hybrid systems, in: Proc. 18th IFAC World Congress, Milan, pp. 12509–12514.

[] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks, EMF Eclipse Modeling Framework, Addison-Wesley, 2009.

[] T. E. Foundation, Eclipse, `http://www.eclipse.org/`, 2011.

[] Systems Engineering, The Compositional Interchange Format for Hybrid Systems, `http://se.wtb.tue.nl/sewiki/cif/start`, 2010.

[] D. E. Nadales Agut, M. A. Reniers, Deriving a simulator for a hybrid language using SOS rules, 2011. Http://se.wtb.tue.nl/sewiki/cif/publications2.

[] W3C, W3C SVG working group, `http://www.w3.org/Graphics/SVG/`, 2011.

[] T. A. Henzinger, The theory of hybrid automata, in: M. K. Inan, R. P. Kurshan (Eds.), Verification of Digital and Hybrid Systems, volume 170 of *NATO ASI Series F: Computer and Systems Science*, Springer-Verlag, New York, 2000, pp. 265–292.

[] P. J. L. Cuijpers, M. A. Reniers, Lost in translation: Hybrid-time flows vs real-time transitions, in: HSCC 2008, volume 4981 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 116–129.

[] N. Lynch, R. Segala, F. Vaandrager, Hybrid I/O automata revisited, in: Proceedings Fourth International Workshop on Hybrid Systems: Computation and Control (HSCC'01), Springer-Verlag, 2001, pp. 403–417.

[] J. C. Reynolds, Theories of programming languages, Cambridge University Press, New York, NY, USA, 1999.

[] P. J. L. Cuijpers, M. A. Reniers, W. P. M. H. Heemels, Hybrid Transition Systems, Technical Report CS-Report 02-12, TU/e, Eindhoven, Netherlands, 2002.

[] D. A. van Beek, P. J. L. Cuijpers, J. Markovski, D. E. Nadales Agut, J. E. Rooda, Reconciling urgency and variable abstraction in a hybrid compositional setting, in: K. Chatterjee, T. Henzinger (Eds.), Formal Modeling and Analysis of Timed Systems, volume 6246 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2010, pp. 47–61.

[] D. A. v. Beek, K. L. Man, M. A. Reniers, J. E. Rooda, R. R. H. Schiffelers, Syntax and Consistent Equation Semantics of Hybrid Chi, Technical Report CS-Report 04-37, Eindhoven University of Technology, Department of Computer Science, The Netherlands, 2004.

[] D. A. v. Beek, A. T. Hofkamp, M. A. Reniers, J. E. Rooda, R. R. H. Schiffelers, Syntax and formal semantics of Chi 2.0, SE Report 2008-01, Eindhoven University of Technology, Systems Engineering Group, Department of Mechanical Engineering, Eindhoven, The Netherlands, 2008.

[] S. Bornot, J. Sifakis, An algebraic framework for urgency, Information and Computation 163 (2000) 172–202.

[] D. S. Scott, Outline of a Mathematical Theory of Computation, Technical Report PRG–2, Oxford University, Oxford, England, 1970.

[] J. Bengtsson, W. Yi, Timed automata: Semantics, algorithms and tools, in: J. Desel, W. Reisig, G. Rozenberg (Eds.), Lectures on Concurrency and Petri Nets, volume 3098 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2004, pp. 87–124.

[] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, O. Maler, SpaceEx: Scalable verification of hybrid systems, in: Proc. 23rd International Conference on Computer Aided Verification (CAV), LNCS, Springer, 2011.

[] D. A. v. Beek, K. L. Man, M. A. Reniers, J. E. Rooda, R. R. H. Schiffelers, Syntax and consistent equation semantics of hybrid Chi, Journal of Logic and Algebraic Programming 68 (2006) 129–210.

[] D. E. Nadales Agut, D. A. van Beek, J. C. M. Baeten, Deliverable D1.2.1 Report on the extension of Chi, Technical Report, MULTIFORM consortium, 2011.

[] D. Nadales Agut, M. A. Reniers, Linearization of CIF through SOS, in: B. Luttik, F. Valencia (Eds.), EXPRESS, volume 64 of *EPTCS*, pp. 74–88.

[] T. P. Engels, CIF to CIF model transformations, Master's thesis, Eindhoven University of Technology, 2010.

[] P. J. L. Cuijpers, Hybrid Process Algebra, Ph.D. thesis, Eindhoven University of Technology, 2004.

[] J. C. M. Baeten, T. Basten, M. A. Reniers, Process Algebra: Equational Theories of Communicating Processes (Cambridge Tracts in Theoretical Computer Science), Cambridge University Press, 1st edition, 2009.

[] H. Bohnenkamp, P. D'Argenio, H. Hermanns, J. Katoen, Modest: A compositional modeling formalism for real-time and stochastic systems, IEEE Transactions on Software Engineering 32 (2006) 812–830.

[] S. Strubbe, R. Langerak, A composition operator for systems with active and passive actions, in: Proceedings of the 25th IFIP WG 6.1 international conference on Formal Techniques for Networked and Distributed Systems, FORTE'05, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 24–37.

Appendix A. Proofs of the hybrid transition systems properties

Proof of Property **??**.
The property that for all p, p', σ, σ', and A we have:

$$(p, \sigma) \overset{A}{\dashrightarrow} (p', \sigma') \Rightarrow \sigma \models p \land \sigma' \models p'$$

can be easily proved by structural induction on the CIF compositions, using the definition of consistency (Definition **??**), and the rules for environment transition. This is shown below.

Basis. This case follows directly from Rule **??** and definition of consistency (Definition **??**).

Induction step. We do a case analysis depending on the structure of composition p.

Parallel composition Let $p \equiv q \parallel r$. Assume:

$$(q \parallel r, \sigma) \overset{A}{\dashrightarrow} (q' \parallel r', \sigma)$$

Inspecting a the environment rules for parallel composition, we have that only Rule **??** could be applied to obtain this transition. Then we have:

$$\begin{aligned} &A \equiv A_q \cup A_r \\ &(q, \sigma) \overset{A_q}{\dashrightarrow} (q', \sigma') \\ &(r, \sigma) \overset{A_r}{\dashrightarrow} (r', \sigma') \end{aligned}$$

By induction hypothesis we get:

$$\begin{aligned} &\sigma \models q \text{ and } \sigma' \models q' \\ &\sigma \models r \text{ and } \sigma' \models r' \end{aligned}$$

And by definition of consistency we get:

$$\sigma \models q \parallel r \text{ and } \sigma \models q' \parallel r'$$

Variable scope Let $p \equiv |[_{\mathrm{V}}\, x = e_0, \dot{x} = e_1 :: q\,]|$. Assume:

$$(|[_{\mathrm{V}}\, x = e_0, \dot{x} = e_1 :: q\,]|, \sigma) \overset{A}{\dashrightarrow} (|[_{\mathrm{V}}\, x = d_0', \dot{x} = d_1' :: q'\,]|, \sigma')$$

By inspecting at the environment rules for variable scope, we know that Rule **??** was applied to obtain the transition above. Thus we have that for some d_0, d_1:

$$(\sigma \models e_0 = d_0) \vee e_0 = \bot$$
$$(\sigma \models e_1 = d_1) \vee e_1 = \bot$$
$$(q, \{x \mapsto d_0, \dot{x} \mapsto d_1\} \succ \sigma) \overset{A}{\dashrightarrow} (q', \{x \mapsto d_0', \dot{x} \mapsto d_1'\} \succ \sigma') \tag{A.1}$$

By induction hypothesis we know that:

$$\{x \mapsto d_0, \dot{x} \mapsto d_1\} \succ \sigma \models q$$
$$\{x \mapsto d_0', \dot{x} \mapsto d_1'\} \succ \sigma' \models q' \tag{A.2}$$

By (**??**) and (**??**), and definition of consistency (Definition **??**) we get:

$$\sigma \models |[_{\mathrm{V}}\, x = e_0, \dot{x} = e_1 :: q\,]| \text{ and } \sigma' \models |[_{\mathrm{V}}\, x = d_0', \dot{x} = d_1' :: q'\,]|$$

which concludes the proof for this case.

The proofs remaining cases can be obtained straightforwardly in a similar manner.

□

Proof of Property **??**. The proof of this property is straightforward, since the definition of the set of synchronizing actions of a composition (Definition **??**) is analogous to the construction of the set of synchronizing actions in the labels of the environment transitions. □

Proof of Property **??**. The proof goes via structural induction on the CIF compositions.

Basis The composition at the target state of an environment transition for an automaton has the form:

$$p' \equiv (V, \mathrm{id}_v, \mathrm{inv}, \mathrm{tcp}, E, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype})$$

since id_v evaluates to true only for location v, by looking at the rule for environment transitions of automata (Rule **??**), the only possible outgoing transitions from state

$$(p', \sigma)$$

has p' as the only possible composition at the destination state, which concludes the proof for this case.

Induction For the parallel composition case, assume $p \equiv q \parallel r$, and that there are two environment transitions:

$$(q \parallel r, \sigma) \overset{A}{\dashrightarrow} (q' \parallel r', \sigma') \text{ and } (q' \parallel r', \varsigma) \overset{B}{\dashrightarrow} (q'' \parallel r'', \varsigma')$$

By using the rule for environment transitions of parallel composition (Rule **??**), and induction hypothesis we get:

$$p' \equiv p'' \text{ and } q' \equiv q''$$

and the result follows from the definition of syntactic equivalence.

For the variable scope case we note that if e_0 and e_1 are values, then e_0 and e_1 evaluate to itself, and we have:

$$(|[_{\mathrm{V}}\, x = e_0, \dot{x} = e_1 :: p\ ]|, \sigma) \overset{A}{\dashrightarrow} (|[_{\mathrm{V}}\, x = e_0, \dot{x} = e_1 :: p'\ ]|, \sigma')$$

for all σ, σ'. Then we can apply the same strategy we used for the parallel composition case.

The cases for the remaining operators are easy to prove in a similar way.

$\square$

Proof of Property **??**. The property can be easily proved by structural induction, and it follows from the time rule for automata, and the fact that the operations on trajectories (such as $\cup$) do not alter their domain.

Basis. Let $p \equiv (V, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, E, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype})$. Assume:

$$(p, \sigma) \overset{\rho, A, \theta}{\longmapsto} (p', \sigma')$$

By inspecting at the time rules for automata, we have that only Rule **??** can be applied to obtain the transition above, thus we get that for some t:

$$\mathrm{dom}(\rho) = [0, t] = \mathrm{dom}(\theta)$$
$$\sigma = \rho(0) \text{ and } \sigma' = \rho(t)$$

which is what we wanted to prove.

Induction step. We do a case analysis depending on the structure of composition p.

Parallel composition Let $p \equiv q \parallel r$. Assume:

$$(p, \sigma) \overset{\rho, A, \theta}{\longmapsto} (p', \sigma')$$

By inspecting at the time rules for parallel composition, we have that necessarily Rule **??** was applied last, thus for some A_q, A_r, θ_q, θ_r:

$$\begin{aligned}
&A \equiv A_q \cup A_r \\
&\theta \equiv (\theta_q \cap \theta_r) \cup (\theta_q \setminus A_r) \cup (\theta_r \setminus A_q) \\
&(q, \sigma) \overset{\rho, A_q, \theta_q}{\longmapsto} (q', \sigma') \text{ and } (r, \sigma) \overset{\rho, A_r, \theta_r}{\longmapsto} (r', \sigma')
\end{aligned}$$

By induction hypothesis, we know that for some t:

$$\begin{aligned}
&\mathrm{dom}(\rho) = [0, t] = \mathrm{dom}(\theta_q) \\
&\rho(0) = \sigma \text{ and } \rho(t) = \sigma'
\end{aligned}$$

Also using induction hypothesis, we know that for some t':

$$\begin{aligned}
&\mathrm{dom}(\rho) = [0, t'] = \mathrm{dom}(\theta_r) \\
&\rho(0) = \sigma \text{ and } \rho(t') = \sigma'
\end{aligned}$$

Then using predicate calculus, we have that necessarily $t = t'$, and since the operators $\cap$, $\setminus$, and $\cup$ do not change the trajectories on which they operate, we have:

$$\mathrm{dom}(\theta) = [0, t]$$

which concludes the proof for this case.

Variable scope and urgency case These cases are similar to the previous case. They follow by applying induction hypothesis, inspecting at the corresponding rules, and the fact that the overwriting and the set difference ($\setminus$) operators do not change their operands' domain.

The remaining cases are straightforward to prove using induction hypothesis, since the labels of the time transitions are propagated unaltered from the premises to the conclusions. □

Proof of Prop **??**. The proof goes via structural induction.

Base Case Assume:

$$\begin{aligned}
&((V, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, E, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype}), \rho(0)) \overset{\rho, \mathrm{act}_S, \theta}{\longmapsto} \\
&((V, \mathrm{id}_v, \mathrm{inv}, \mathrm{tcp}, E, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype}), \rho(t))
\end{aligned}$$

By inspecting the time rule for automata we know that:

$$\begin{aligned}&\mathrm{dom}(\rho) = [0,t], 0 < t, \mathrm{dom}(\rho) = \mathrm{dom}(\theta), \rho(0) \models \mathrm{init}(v),\\ &\langle \forall s : s \in \mathrm{dom}(\theta) : \theta(s) = \{a \mid (v,g,a,u,v') \in E \wedge \rho(s) \models g\}\rangle,\\ &\langle \forall s : s \in [0,t] : \rho(s) \models \mathrm{inv}(v)\rangle, \langle \forall s : s \in [0,t) : \rho(s) \models \mathrm{tcp}(v)\rangle,\\ &\langle \forall x : x \in \mathrm{dom}(\mathrm{dtype}) : (\rho_x, \rho_{\dot{x}}) \in \mathrm{dtype}(x)\rangle\end{aligned}$$

Using predicate calculus, the definition of the trajectory prefix operator, and the fact that

$$(\rho_x, \rho_{\dot{x}}) \in G \Rightarrow (\rho_x^{\leq s}, \rho_{\dot{x}}^{\leq s}) \in G$$

for all dynamic types G and for all s such that $0 < s$, we arrive to the desired result.

Inductive Step

Variable Scope Assume:

$$(\llbracket_{\mathrm{V}}\, x = e_0, \dot{x} = e_1 :: p \rrbracket, \rho(0)) \overset{\rho, A, \theta}{\longmapsto} (\llbracket_{\mathrm{V}}\, x = d_0', \dot{x} = d_1' :: p' \rrbracket, \rho(t))$$

Then, inspecting the rule for variable scope we know that:

$$\begin{aligned}&(\sigma \models e_0 = d_0) \vee e_0 = \bot, (\sigma \models e_1 = d_1) \vee e_1 = \bot,\\ &(p, \{x \mapsto d_0, \dot{x} \mapsto d_1\} \succ \sigma) \overset{\{x \mapsto f_0, \dot{x} \mapsto f_1\} \succ \rho, A, \theta}{\longmapsto}\\ &(p', \{x \mapsto d_0', \dot{x} \mapsto d_1'\} \succ \sigma')\end{aligned}$$

By induction hypothesis we get:

$$\begin{aligned}&(p, \{x \mapsto d_0, \dot{x} \mapsto d_1\} \succ \sigma) \overset{(\{x \mapsto f_0, \dot{x} \mapsto f_1\} \succ \rho)^{\leq s}, A, \theta^{\leq s}}{\longmapsto}\\ &(p', \{x \mapsto d_0'', \dot{x} \mapsto d_1''\} \succ \sigma'')\end{aligned}$$

Using the definition of trajectory prefix, the fact that

$$(\{x \mapsto f_0, \dot{x} \mapsto f_1\} \succ \rho)^{\leq s} = (\{x \mapsto f_0, \dot{x} \mapsto f_1\})^{\leq s} \succ \rho^{\leq s}$$

and the rule for variable scope we obtain:

$$\begin{aligned}&(\llbracket_{\mathrm{V}}\, x = e_0, \dot{x} = e_1 :: p \rrbracket, \rho(0)) \overset{\rho^{\leq s}, A, \theta^{\leq s}}{\longmapsto}\\ &(\llbracket_{\mathrm{V}}\, x = d_0'', \dot{x} = d_1'' :: p' \rrbracket, \rho^{\leq s}(s))\end{aligned}$$

Other Cases The proof for the remaining operators is straightforward using the SOS rules for time. Only for the dynamic type operator we need the property that:

$$(\rho_x, \rho_{\dot{x}}) \in G \Rightarrow (\rho_{\overline{x}}^{\leq s}, \rho_{\dot{\overline{x}}}^{\leq s}) \in G$$

for all dynamic types G and for all s such that $0 < s$.

□

Proof of Property **??**. The proof is analogous to the proof of Property **??**. In this case we need to use the fact that:

$$(\rho_x, \rho_{\dot{x}}) \in G \Rightarrow (\rho_{\overline{x}}^{\geq s}, \rho_{\dot{\overline{x}}}^{\geq s}) \in G$$

for all dynamic types G and for all s such that $s < t$. □

Proof of Property **??**. The proof goes via structural induction on the CIF compositions.

Basis Assume $p \equiv (V, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, E, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype})$, and assume:

$$(p, \sigma) \overset{\rho,A,\theta}{\longmapsto} (p', \sigma') \text{ and } (p', \sigma') \overset{\rho',A,\theta'}{\longmapsto} (p'', \sigma'')$$

Then, inspecting the time rule for automata we know that there must be some $v \in V$, such that the following conditions hold:

$\mathrm{dom}(\rho) = [0, t] = \mathrm{dom}(\theta), 0 < t$
$\rho(0) \models \mathrm{init}(v)$
$\langle \forall s : s \in \mathrm{dom}(\theta) : \theta(s) = \{a | (v, g, a, u, v') \in E \wedge \rho(s) \models g\} \rangle$
$\langle \forall s : s \in [0, t] : \rho(s) \models \mathrm{inv}(v) \rangle, \langle \forall s : s \in [0, t) : \rho(s) \models \mathrm{tcp}(v) \rangle$
$\langle \forall x : x \in \mathrm{dom}(\mathrm{dtype}) : (\rho_x, \rho_{\dot{x}}) \in \mathrm{dtype}(x) \rangle$
$\mathrm{dom}(\rho') = [0, t'] = \mathrm{dom}(\theta'), 0 < t'$
$\rho'(0) \models \mathrm{id}_v(v)$
$\langle \forall s : s \in \mathrm{dom}(\theta') : \theta'(s) = \{a | (v, g, a, u, v') \in E \wedge \rho'(s) \models g\} \rangle$
$\langle \forall s : s \in [0, t'] : \rho'(s) \models \mathrm{inv}(v) \rangle, \langle \forall s : s \in [0, t') : \rho'(s) \models \mathrm{tcp}(v) \rangle$
$\langle \forall x : x \in \mathrm{dom}(\mathrm{dtype}) : (\rho'_x, \rho'_{\dot{x}}) \in \mathrm{dtype}(x) \rangle$
$\sigma' = \rho(t) = \rho'(0)$
$\rho'(t) = \sigma''$

Using the previous facts, the fact that $\rho(t) = \rho'(0)$, the definition of the concatenation operator for flows, predicate calculus, and algebra, we get that the following conditions hold.

$$\begin{array}{l}
\mathrm{dom}(\rho \cdot_t \rho') = [0, t + t'] = \mathrm{dom}(\theta \cdot_t \theta'), 0 < t + t' \\
(\rho \cdot_t \rho')(0) \models \mathrm{init}(v) \\
\langle \forall s : s \in \mathrm{dom}(\theta \cdot_t \theta') : (\theta \cdot_t \theta')(s) = \{a \mid (v, g, a, u, v') \in E \land (\rho \cdot_t \rho')(s) \models g\}\rangle \\
\langle \forall s : s \in [0, t] : (\rho \cdot_t \rho')(s) \models \mathrm{inv}(v)\rangle, \langle \forall s : s \in [0, t) : (\rho \cdot_t \rho')(s) \models \mathrm{tcp}(v)\rangle
\end{array}$$

Using the property of dynamic types and the concatenation operator we also obtain

$$\langle \forall x : x \in \mathrm{dom}(\mathrm{dtype}) : ((\rho \cdot_t \rho')_x, (\rho \cdot_t \rho')_{\dot{x}}) \in \mathrm{dtype}(x)\rangle$$

Then, using the time rule for automata we conclude that there is a time transition:

$$(p, \sigma) \overset{\rho \cdot_t \rho', A, \theta \cdot_t \theta'}{\longmapsto} (p'', \sigma'')$$

which concludes the proof for this case.

Inductive Step For the parallel composition case assume there are two time transitions

$$\begin{array}{l}
(p \parallel q, \sigma) \overset{\rho, A_p \cup A_q, (\theta_p \cap \theta_q) \cup (\theta_p \setminus A_q) \cup (\theta_q \setminus A_p)}{\longmapsto} (p' \parallel q', \sigma') \\
(p' \parallel q', \sigma') \overset{\rho', A_p \cup A_q, (\theta'_p \cap \theta'_q) \cup (\theta'_p \setminus A_q) \cup (\theta'_q \setminus A_p)}{\longmapsto} (p'' \parallel q'', \sigma'')
\end{array}$$

By inspecting the rules for parallel composition, and using induction hypothesis we get that there are two time transitions:

$$\begin{array}{l}
(p, \sigma) \overset{\rho \cdot_t \rho', A, \theta_p \cdot_t \theta'_p}{\longmapsto} (p'', \sigma'') \\
(q, \sigma) \overset{\rho \cdot_t \rho', A, \theta_q \cdot_t \theta'_q}{\longmapsto} (q'', \sigma'')
\end{array}$$

Using the time rule for parallel composition we get that there is a transition:

$$(p \parallel q, \sigma) \overset{\rho \cdot_t \rho', A, ((\theta_p \cdot_t \theta'_p) \cap (\theta_q \cdot_t \theta'_q)) \cup ((\theta_p \cdot_t \theta'_p) \setminus A_q) \cup ((\theta_q \cdot_t \theta'_q) \setminus A_p)}{\longmapsto} (p'' \parallel q'', \sigma'')$$

Using the definition of the flow concatenation operator it is easy to show that:

$$\begin{array}{l}
((\theta_p \cdot_t \theta'_p) \cap (\theta_q \cdot_t \theta'_q)) \cup ((\theta_p \cdot_t \theta'_p) \setminus A_q) \cup ((\theta_q \cdot_t \theta'_q) \setminus A_p) = \\
((\theta_p \cap \theta_q) \cup (\theta_p \setminus A_q) \cup (\theta_q \setminus A_p)) \cdot_t ((\theta'_p \cap \theta'_q) \cup (\theta'_p \setminus A_q) \cup (\theta'_q \setminus A_p))
\end{array}$$

which concludes the proof for the parallel composition case.

The urgency operator case can be proved using the fact that:

$$\langle \forall s : s \in [0,t) : a \notin \theta(s) \rangle \wedge \langle \forall s : s \in [0,t') : a \notin \theta'(s) \rangle \Rightarrow$$
$$\langle \forall s : s \in [0,t+t') : a \notin (\theta \cdot_t \theta')(s) \rangle$$

For all θ, θ', such that $\mathrm{dom}(\theta) = [0,t]$, $\mathrm{dom}(\theta') = [0,t']$, and $\theta(t) = \theta'(0)$.

The dynamic type case can be proved using the property of dynamic types and the concatenation operator.

For the variable scope case, assume there are two time transitions:

$$(\llbracket_{\mathrm{V}}\, x = d_0, \dot{x} = d_1 :: p \,\rrbracket, \rho(0)) \overset{\rho, A, \theta}{\longmapsto} (\llbracket_{\mathrm{V}}\, x = d_0', \dot{x} = d_1' :: p' \,\rrbracket, \rho(t)) \text{ and}$$
$$(\llbracket_{\mathrm{V}}\, x = d_0', \dot{x} = d_1' :: p' \,\rrbracket, \rho'(0)) \overset{\rho', A, \theta'}{\longmapsto} (\llbracket_{\mathrm{V}}\, x = d_0'', \dot{x} = d_1'' :: p'' \,\rrbracket, \rho'(t'))$$

where to simplify the details of the proof we assume d_0, d_1 are values (the proof can be easily extended to the case when local variables are initialized using expressions).

Using the rule for variable scope, we know that there are two time transitions:

$$(p, \{x \mapsto d_0, \dot{x} \mapsto d_1\} \succ \sigma) \overset{\{x \mapsto f_0, \dot{x} \mapsto f_1\} \succ \rho, A, \theta}{\longmapsto}$$
$$(p', \{x \mapsto d_0', \dot{x} \mapsto d_1'\} \succ \sigma') \text{ and}$$
$$(p', \{x \mapsto d_0', \dot{x} \mapsto d_1'\} \succ \sigma') \overset{\{x \mapsto g_0, \dot{x} \mapsto g_1\} \succ \rho', A, \theta'}{\longmapsto}$$
$$(p'', \{x \mapsto d_0'', \dot{x} \mapsto d_1''\} \succ \sigma'')$$

Using Property **??** we know that

$$(\{x \mapsto f_0, \dot{x} \mapsto f_1\} \succ \rho)(t) = \{x \mapsto d_0, \dot{x} \mapsto d_1\} \succ \sigma =$$
$$(\{x \mapsto g_0, \dot{x} \mapsto g_1\} \succ \rho')(0)$$

then we can apply induction hypothesis to obtain a transition:

$$(p, \{x \mapsto d_0, \dot{x} \mapsto d_1\} \succ \sigma) \overset{(\{x \mapsto f_0, \dot{x} \mapsto f_1\} \succ \rho) \cdot_t (\{x \mapsto g_0, \dot{x} \mapsto g_1\} \succ \rho'), A, \theta \cdot_t \theta'}{\longmapsto}$$
$$(p'', \{x \mapsto d_0'', \dot{x} \mapsto d_1''\} \succ \sigma'')$$

Using the definition of the flow concatenation operator it is possible to prove that:

$$(\{x \mapsto f_0, \dot{x} \mapsto f_1\} \succ \rho) \cdot_t (\{x \mapsto g_0, \dot{x} \mapsto g_1\} \succ \rho') =$$
$$\{x \mapsto f_0 \cdot_t g_0, \dot{x} \mapsto f_1 \cdot_t g_1\} \succ \rho \cdot_t \rho'$$

Then using the time rule for variable scope we arrive to the desired result, which concludes the proof for this case.

For the remaining operators the proof is straightforward.

□

Appendix B. Proof of the CIF operators properties

Proof of Property **??**. First we show that parallel composition is commutative. To this end, we show that the relation

$$R \triangleq \{(p \parallel q, q \parallel p) \mid p \in \mathcal{C}, q \in \mathcal{C}\}$$

is a witness of the bisimulation.

Given a transition of the form:

$$(p \parallel q, \sigma) \xrightarrow{a,b,X} (p' \parallel q', \sigma') \tag{B.1}$$

the fact that there is a corresponding transition

$$(q \parallel p, \sigma) \xrightarrow{a,b,X} (q' \parallel p', \sigma')$$

can be inferred by swapping the premises and using the rule for synchronizing parallel composition, or using the symmetric rule of the interleaving rule that originated transition (**??**).

For time and environment transitions, the symmetric version of these transitions can be obtained by inverting the order of the premises that originated them, and using the fact that the operators $\cap$ and $\cup$ are commutative.

To show that parallel composition is associative, we prove that the relation

$$R \triangleq \{((p \parallel q) \parallel r, p \parallel (q \parallel r)) \mid p \in \mathcal{C}, q \in \mathcal{C}, r \in \mathcal{C}\}$$

is a witness of the bisimulation.

Given an action transition of the form:

$$((p \parallel q) \parallel r, \sigma) \xrightarrow{a,b,X} ((p' \parallel q') \parallel r', \sigma')$$

the existence of a transition

$$(p \parallel (q \parallel r), \sigma') \xrightarrow{a,b,X} (p' \parallel (q' \parallel r'), \sigma')$$

can be inferred by inspecting the rules for parallel composition, and grouping the premises in a different order.

Given a time transition of the form:

$$((p \parallel q) \parallel r, \sigma) \overset{\rho, A, \theta}{\longmapsto} ((p' \parallel q') \parallel r', \sigma')$$

the existence of a transition

$$(p \parallel (q \parallel r), \sigma') \overset{\rho, A, \theta}{\longmapsto} (p' \parallel (q' \parallel r'), \sigma')$$

follows from the associativity of $\cup$, and the fact that for all guard trajectories θ_i and sets of actions A_i, $i \in \{p, q, r\}$, the following equality holds:

$$\begin{aligned}
&(\theta_p \cap ((\theta_q \cap \theta_r) \cup (\theta_q \setminus A_r) \cup (\theta_r \setminus A_q))) \cup \\
&(\theta_p \setminus (A_q \cup A_r)) \cup \\
&(((\theta_q \cap \theta_r) \cup (\theta_q \setminus A_r) \cup (\theta_r \setminus A_q)) \setminus A_p) \\
= \\
&(((\theta_p \cap \theta_q) \cup (\theta_p \setminus A_q) \cup (\theta_q \setminus A_p)) \cap \theta_r) \cup \\
&(((\theta_p \cap \theta_q) \cup (\theta_p \setminus A_q) \cup (\theta_q \setminus A_p)) \setminus A_r) \cup \\
&(\theta_r \setminus (A_p \cup A_q))
\end{aligned}$$

Finally the transfer condition for environment transitions holds due to the associativity property of $\cup$. □

Proof of Property **??**. To prove that

$$\mathrm{D}_{x:G}(p) \parallel q \leftrightarrow \mathrm{D}_{x:G}(p \parallel q)$$

we show that $R \cup R^{-1}$ is a witness of the bisimulation, where:

$$R \triangleq \{(\mathrm{D}_{x:G}(p) \parallel q, \mathrm{D}_{x:G}(p \parallel q)) \mid x \in \mathcal{V}, G \in \mathcal{D}, p \in \mathcal{C}, q \in \mathcal{C}\}.$$

The validity of the transfer conditions for action, time, and environment transitions follow from the fact that the dynamic type operator does not restrict the set of allowed action, time, or environment transitions for a given composition. In particular, for the proof of the transfer conditions for time transitions the predicate $(\rho_x, \rho_{\dot{x}}) \in G$, obtained by inspecting the time rule for dynamic type, can be used to introduce the dynamic type operator where required.

To prove that

$$(u \gg p) \parallel q \leftrightarrow u \gg (p \parallel q)$$

we show that $R \cup R^{-1}$ is a witness of the bisimulation, where:

$$R \triangleq \{((u \gg p) \parallel q, u \gg (p \parallel q)) \mid u \in \mathcal{P}_t, p \in \mathcal{C}, q \in \mathcal{C}\}$$

The proof of validity of the transfer conditions for the three kinds of transitions follows from similar arguments as those given in the previous proof. □

Proof of Prop **??**. There are 22 bisimulation relations to be proved. Most of these proofs can be obtained in a similar way. We present here the most relevant cases.

To prove that

$$\gamma_{a'}(\gamma_a(p)) \underline{\leftrightarrow} \gamma_a(\gamma_{a'}(p))$$

we use as a witness relation

$$R \triangleq \{(\gamma_{a'}(\gamma_a(p)), \gamma_a(\gamma_{a'}(p))) \mid a' \in \mathcal{A}, a \in \mathcal{A}, p \in \mathcal{C}\}$$

The validity of the three transfer conditions is a consequence of the commutativity of $\vee$ and $\cup$.

To prove that

$$[\![_{\mathrm{A}}\, a' :: [\![_{\mathrm{A}}\, a :: p\ ]\!]]\!] \ \underline{\leftrightarrow}\ [\![_{\mathrm{A}}\, a :: [\![_{\mathrm{A}}\, a' :: p\ ]\!]]\!]$$

we use as a witness relation

$$R \triangleq \{([\![_{\mathrm{A}}\, a' :: [\![_{\mathrm{A}}\, a :: p\ ]\!]]\!], [\![_{\mathrm{A}}\, a :: [\![_{\mathrm{A}}\, a' :: p\ ]\!]]\!]) \mid a' \in \mathcal{A}, a \in \mathcal{A}, p \in \mathcal{C}\}$$

The validity of transfer conditions for time and environment transitions follows from the commutativity of $\cap$. The transfer condition for actions requires a case analysis. To prove:

$$\begin{array}{l} \quad ([\![_{\mathrm{A}}\, a' :: [\![_{\mathrm{A}}\, a :: p\ ]\!]]\!], \sigma) \xrightarrow{a''',b,X} ([\![_{\mathrm{A}}\, a' :: [\![_{\mathrm{A}}\, a :: p'\ ]\!]]\!], \sigma') \\ \Rightarrow \\ \quad ([\![_{\mathrm{A}}\, a :: [\![_{\mathrm{A}}\, a' :: p\ ]\!]]\!], \sigma) \xrightarrow{a''',b,X} ([\![_{\mathrm{A}}\, a :: [\![_{\mathrm{A}}\, a' :: p'\ ]\!]]\!], \sigma') \end{array}$$

we note first that, according to the rules for action scope operator, there must be an action transition:

$$(p, \sigma) \xrightarrow{a'',b',X} (p', \sigma')$$

for some $a'' \in \mathcal{A}_\tau$, $b' \in \mathbb{B}$.

Then, we need to consider 4 cases:

Case $a = a' = a''$ Then, according to Rule **??**, $a''' \equiv \tau$, and $b \equiv \mathrm{false}$. If we invert the order of application of the scope operators, then the action rule for variable scope ensures that then same labels will result in the conclusion.

Case $a = a''$ **and** $a' \neq a''$ Then, as in the previous case, by Rule **??**, $a''' \equiv \tau$, and $b \equiv \mathrm{false}$.

Now consider

$$
\begin{aligned}
&(p,\sigma) \xrightarrow{a'',b',X} (p',\sigma')\\
&\Leftrightarrow \{ \text{ Rule } \textbf{??},\ a' \neq a'' \ \}\\
&(\llbracket_{\mathrm{A}}\, a' :: p \,\rrbracket, \sigma) \xrightarrow{a'',b',X} (\llbracket_{\mathrm{A}}\, a' :: p' \,\rrbracket, \sigma')\\
&\Leftrightarrow \{ \text{ Rule } \textbf{??},\ a = a'' \ \}\\
&(\llbracket_{\mathrm{A}}\, a :: \llbracket_{\mathrm{A}}\, a' :: p \,\rrbracket\rrbracket, \sigma) \xrightarrow{\tau,\mathrm{false},X} (\llbracket_{\mathrm{A}}\, a :: \llbracket_{\mathrm{A}}\, a' :: p' \,\rrbracket\rrbracket, \sigma')
\end{aligned}
$$

Case $a' = a''$ **and** $a \neq a''$ Similar to the previous case.

Case $a \neq a''$ **and** $a' \neq a''$ Then, $a'' = a'''$, and according to the action rule for variable scope, the label is not affected, and therefore the order of application of the operator can be inverted without altering the resulting label.

The fact that

$$
\begin{aligned}
&(\llbracket_{\mathrm{A}}\, a' :: \llbracket_{\mathrm{A}}\, a :: p \,\rrbracket\rrbracket, \sigma) \xrightarrow{a'',b,X} (\llbracket_{\mathrm{A}}\, a' :: \llbracket_{\mathrm{A}}\, a :: p' \,\rrbracket\rrbracket, \sigma')\\
\Leftarrow&\\
&(\llbracket_{\mathrm{A}}\, a :: \llbracket_{\mathrm{A}}\, a' :: p \,\rrbracket\rrbracket, \sigma) \xrightarrow{a'',b,X} (\llbracket_{\mathrm{A}}\, a :: \llbracket_{\mathrm{A}}\, a' :: p' \,\rrbracket\rrbracket, \sigma')
\end{aligned}
$$

can be proven in a symmetric way.

The validity of the remaining properties is a consequence of the non-interference between the side conditions of the operators and the change they cause on the labels. Next, we give an example of this by proving

$$\gamma_a(u \gg p) \underline{\leftrightarrow} u \gg (\gamma_a(p))$$

The above equivalence can be proven using as a witness $R \cup R^{-1} \cup I_{\mathcal{C}}$, where

$$R \triangleq \{(\gamma_a(u \gg p), u \gg (\gamma_a(p))) \mid a \in \mathcal{A}, u \in \mathcal{P}_t, p \in \mathcal{C}\}$$

and $I_{\mathcal{C}}$ is the identity function on compositions.

Next we prove the validity of the transfer condition for action transitions. To this end we make the following derivation:

$$
\begin{aligned}
&(\gamma_a(u \gg p), \sigma) \xrightarrow{a',b \vee a=a',X} (\gamma_a(p'), \sigma')\\
&\Leftrightarrow \{ \text{ Rule for synchronization } \}\\
&(u \gg p, \sigma) \xrightarrow{a',b,X} (p', \sigma')\\
&\Leftrightarrow \{ \text{ Rule for initialization } \}
\end{aligned}
$$

$(p, \sigma) \xrightarrow{a', b, X} (p', \sigma'), \sigma \models u$

$\Leftrightarrow$ { Rule for synchronization }

$(\gamma_a(p), \sigma) \xrightarrow{a', b \vee a = a', X} (\gamma_a(p'), \sigma'), \sigma \models u$

$\Leftrightarrow$ { Rule for initialization }

$(u \gg \gamma_a(p), \sigma) \xrightarrow{a', b \vee a = a', X} (\gamma_a(p'), \sigma')$

The validity of the remaining transfer conditions is a consequence of the fact that rules for initialization operator and synchronizing actions do not interfere in the changes they cause to the labels, or in the side conditions they require. $\square$

Proof of Prop **??**. To prove

$$\gamma_a(\gamma_a(p)) \underline{\leftrightarrow} \gamma_a(p)$$

we use as a witness relation $R \cup R^{-1}$, where

$$R \triangleq \{(\gamma_a(\gamma_a(p)), \gamma_a(p)) \mid a \in \mathcal{A}, p \in \mathcal{C}\}.$$

Then the validity of the transfer conditions follows from the idempotency of $\vee$ and $\cup$.

The validity of

$$u \gg (u \gg p)$$

follows from Property **??** and idempotency of $\wedge$.

To prove the validity of

$$[\![_{\mathrm{V}}\, a :: [\![_{\mathrm{V}}\, a :: p\,]\!]]\!] \;\underline{\leftrightarrow}\; [\![_{\mathrm{V}}\, a :: p\,]\!]$$

we use as a witness relation $R \cup R^{-1}$, where

$$R \triangleq \{([\![_{\mathrm{V}}\, a :: [\![_{\mathrm{V}}\, a :: p\,]\!]]\!], [\![_{\mathrm{V}}\, a :: p\,]\!]) \mid a \in \mathcal{A}, p \in \mathcal{C}\}$$

The validity of the transfer conditions for time and environment transitions follows from the idempotency of $\cap$. For action transitions, a proof can be obtained by performing a simple case analysis.

To prove the validity of

$$\upsilon_a(\upsilon_a(p)) \underline{\leftrightarrow} \upsilon_a(p)$$

we use as a witness relation $R \cup R^{-1}$, where

$$R \triangleq \{(\upsilon_a(\upsilon_a(p)), \upsilon_a(p)) \mid a \in \mathcal{A}, p \in \mathcal{C}\}$$

The validity of the transfer conditions for action and environment transitions follows from the fact that the urgency operator does not affect these kind of transitions. For time transitions, the validity of the transfer conditions follows from the idempotency of entailment (condition $\langle \forall s : s \in [0,t) : a \notin \theta(s) \rangle$).

The validity of

$$\mathrm{D}_{x:G}(\mathrm{D}_{x:G}(p)) \underline{\leftrightarrow} \mathrm{D}_{x:G}(p)$$

follows from Prop **??** and idempotency of $\cap$.

To prove the validity of

$$\mathrm{ctrl}_x(\mathrm{ctrl}_x(p)) \underline{\leftrightarrow} \mathrm{ctrl}_x(p)$$

we use as a witness relation $R \cup R^{-1}$, where

$$R \triangleq \{(\mathrm{ctrl}_x(\mathrm{ctrl}_x(p)), \mathrm{ctrl}_x(p)) \mid x \in \mathcal{V}, p \in \mathcal{C}\}$$

The validity of the transfer conditions follows from the fact that the control variable operator does not affect time and environment transitions, idempotency of $\cup$, and idempotency of entailment. $\square$

Proof of Prop **??**. To prove

$$u \gg (v \gg p) \underline{\leftrightarrow} (u \wedge v) \gg p$$

we use as a witness relation $R \cup R^{-1} \cup I_{\mathcal{C}}$, where

$$R \triangleq \{(u \gg (v \gg p), (u \wedge v) \gg p) \mid u \in \mathcal{P}_t, v \in \mathcal{P}_t, p \in \mathcal{C}\}$$

The validity of the transfer conditions is a consequence of the validity of

$$(\sigma \models u, \sigma \models v) \Leftrightarrow (\sigma \models u \wedge v)$$

for the satisfaction relation.

We illustrate the proof of the transfer condition for action transitions. The proofs for the remaining cases are analogous.

We make the following derivation. Assume:

$(u \gg (v \gg p), \sigma) \xrightarrow{a,b,X} (p', \sigma')$

$\Leftrightarrow$ { Only Rule **??** cab be applied}

$(v \gg p, \sigma) \xrightarrow{a,b,X} (p', \sigma'), \sigma \models u$

$\Leftrightarrow$ { Only Rule **??** cab be applied}

$(p, \sigma) \xrightarrow{a,b,X} (p', \sigma'), \sigma \models u, \sigma \models v$

$\Leftrightarrow$ { Definition of satisfaction relation ($\models$) conjunctions}

$(p, \sigma) \xrightarrow{a,b,X} (p', \sigma'), \sigma \models u \wedge v$

$\Leftrightarrow$ { Only Rule **??** cab be applied}

$(u \wedge v \gg p, \sigma) \xrightarrow{a,b,X} (p', \sigma')$

$\square$

Proof of Property **??**. To prove

$$\mathrm{D}_{x:G_0}(\mathrm{D}_{x:G_1}(p)) \underline{\leftrightarrow} \mathrm{D}_{x:G_0 \cap G_1}(p)$$

we use as witness relation $R \cup R^{-1}$, where

$$R \triangleq \{(\mathrm{D}_{x:G_0}(\mathrm{D}_{x:G_1}(p)), \mathrm{D}_{x:G_0 \cap G_1}(p)) \mid x \in \mathcal{V}, G_0 \in \mathcal{D}, G_1 \in \mathcal{D}, p \in \mathcal{C}\}$$

Next, we prove the validity of the transfer condition for time. To this, we make the following derivation:

$(\mathrm{D}_{x:G_0}(\mathrm{D}_{x:G_1}(p)), \sigma) \overset{\rho,A,\theta}{\longmapsto} (\mathrm{D}_{x:G_0}(\mathrm{D}_{x:G_1}(p')), \sigma')$

$\Leftrightarrow$ { Rule for dynamic type operator, twice}

$(p, \sigma) \overset{\rho,A,\theta}{\longmapsto} (p', \sigma'), (\rho_x, \rho_{\dot{x}}) \in G_0, (\rho_x, \rho_{\dot{x}}) \in G_1$

$\Leftrightarrow$ { Set algebra }

$(p, \sigma) \overset{\rho,A,\theta}{\longmapsto} (p', \sigma'), (\rho_x, \rho_{\dot{x}}) \in G_0 \cap G_1$

$\Leftrightarrow$ { Rule for dynamic type operator }

$(\mathrm{D}_{x:G_0 \cap C_0}(p), \sigma) \overset{\rho,A,\theta}{\longmapsto} (\mathrm{D}_{x:G_0 \cap G_1}(p'), \sigma')$

The transfer conditions for action and environment transitions follow from the fact that the dynamic type operator does not restrict these transitions. $\square$

Proof of Prop **??**. We sketch the proof of commutativity of variable scope with regard to parallel composition. The remaining commutativity laws can be proven in a similar way.

To prove

$$[\![_{\mathrm{V}}\, x = e_0, \dot{x} = e_1 :: p\,]\!] \parallel q \underline{\leftrightarrow} [\![_{\mathrm{V}}\, (x = e_0, \dot{x} = e_1)[y, \dot{y}/x, \dot{x}] :: p[y, \dot{y}/x, \dot{x}] \parallel q\,]\!]$$

we use as witness relation $R \cup R^{-1}$, where

$$\begin{aligned} R \triangleq \{ (& [\![_{\mathrm{V}}\, x = e_0, \dot{x} = e_1 :: p\,]\!] \parallel q, \\ & [\![_{\mathrm{V}}\, (x = e_0, \dot{x} = e_1)[y, \dot{y}/x, \dot{x}] :: p[y, \dot{y}/x, \dot{x}] \parallel q\,]\!]) \mid \\ & x \in \mathcal{V}, y \in \mathcal{V}, y \notin \mathrm{vars}(p), \dot{y} \notin \mathrm{vars}(p), e_0 \in \mathcal{E}, e_1 \in \mathcal{E}, p \in \mathcal{C}, q \in \mathcal{C} \} \end{aligned}$$

and $\mathrm{vars}(p)$ refers to the set of variables appearing in composition p.

The fact that $R \cup R^{-1}$ is indeed a witness relation is a consequence of the following lemmas.

Lemma 1 (Variable exchange in action transitions)**.** *For all y, p, x, X, u, u', v, v', w, w', σ, σ', a, and b, such that $y \notin \mathrm{vars}(p)$, we have:*

$$\begin{aligned} & (x \in X \Rightarrow u = u') \wedge \\ & (p, \{x \mapsto v\} \succ \{x \mapsto u\} \succ \{y \mapsto w\} \succ \sigma) \xrightarrow{a,b,X\setminus\{x\}} \\ & (p', \{x \mapsto v'\} \succ \{x \mapsto u'\} \succ \{y \mapsto w'\} \succ \sigma') \end{aligned}$$

if and only if:

$$\begin{aligned} & (y \in X \Rightarrow w = w') \wedge \\ & (p[y/x], \{y \mapsto v\} \succ \{x \mapsto u\} \succ \{y \mapsto w\} \succ \sigma) \xrightarrow{a,b,X\setminus\{y\}} \\ & (p'[y/x], \{y \mapsto v'\} \succ \{x \mapsto u'\} \succ \{y \mapsto w'\} \succ \sigma') \end{aligned}$$

Lemma 2 (Variable overwriting in action transitions)**.** *For all p, p', y, σ, σ', a, b, X, v, and v' such that $y \notin \mathrm{vars}(p)$, we have:*

$$(p, \sigma) \xrightarrow{a,b,X} (p', \sigma')$$

if and only if

$$(p, \{y \mapsto v\} \succ \sigma) \xrightarrow{a,b,X\setminus\{y\}} (p', \{y \mapsto v'\} \succ \sigma')$$

Lemma 3 (Variable exchange in environment transitions)**.** *For all y, p, x, v, v', σ, σ', and A, such that $y \notin \mathrm{vars}(p)$, we have:*

$$(p, \{x \mapsto v\} \succ \sigma) \overset{A}{\dashrightarrow} (p', \{x \mapsto v'\} \succ \sigma')$$

if and only if:

$$(p[y/x], \{y \mapsto v\} \succ \sigma) \overset{A}{\dashrightarrow} (p'[y/x], \{y \mapsto v'\} \succ \sigma')$$

Lemma 4 (Variable overwriting in environment transitions)**.** *For all p, p', y, σ, σ', A, v, and v' such that $y \notin \mathrm{vars}(p)$, we have:*

$$(p, \sigma) \overset{A}{\dashrightarrow} (p', \sigma')$$

if and only if

$$(p, \{y \mapsto v\} \succ \sigma) \overset{A}{\dashrightarrow} (p', \{y \mapsto v'\} \succ \sigma')$$

Lemma 5 (Variable exchange in time transitions)**.** *For all y, p, x, v, v', f σ, σ', ρ, θ, and A, such that $y \notin \mathrm{vars}(p)$, we have:*

$$(p, \{x \mapsto v\} \succ \sigma) \overset{\{x \mapsto f\} \succ \rho, A}{\longmapsto} (p', \{x \mapsto v'\} \succ \sigma')$$

if and only if:

$$(p[y/x], \{y \mapsto v\} \succ \sigma) \overset{\{y \mapsto f\} \succ \rho, A}{\longmapsto} (p'[y/x], \{y \mapsto v'\} \succ \sigma')$$

Lemma 6 (Variable overwriting in environment transitions)**.** *For all p, p', y, σ, σ', ρ, θ, A, v, and v' such that $y \notin \mathrm{vars}(p)$, we have:*

$$(p, \sigma) \overset{\rho, A, \theta}{\longmapsto} (p', \sigma')$$

if and only if

$$(p, \{y \mapsto v\} \succ \sigma) \overset{\{y \mapsto f\} \succ \rho, A}{\longmapsto} (p', \{y \mapsto v'\} \succ \sigma')$$

These lemmas can be proved by structural induction, the fact that if $y \notin \mathrm{vars}(e)$, then

$$(\sigma \models u) \Leftrightarrow (\{y \mapsto v\} \succ \sigma) \models u$$

and the fact that if $y \notin \mathrm{vars}(e)$, then

$$(\{x \mapsto v\} \succ \sigma \models u) \Leftrightarrow (\{y \mapsto v\} \succ \sigma) \models u[y/x]$$

□

For proving Property **??** we need to make use of the following lemmas.

Lemma 7. *For all p, σ, a, s, X, p', σ', and c, such that c does not appear in p, we have:*

$$(p, \sigma) \xrightarrow{a,s,X} (p', \sigma')$$

if and only if

$$(p[c/b], \sigma) \xrightarrow{a[c/b],s,X} (p'[c/b], \sigma')$$

Lemma 8. *For all p, σ, A, p', σ', and c, such that c does not appear in p, we have:*

$$(p, \sigma) \overset{A}{\dashrightarrow} (p', \sigma')$$

if and only if

$$(p[c/b], \sigma) \overset{A[c/b]}{\dashrightarrow} (p'[c/b], \sigma')$$

Given a set A and two elements b and c, the substitution of b by c in A is defined as follows:

$$A[c/b] = \begin{cases} A & \text{if } b \notin A \\ (A \setminus \{b\}) \cup \{c\} & \text{if } b \in A \end{cases}$$

For time transition a similar lemma is needed. We omit it here to since it conveys no theoretical interest.

Next, we give the proofs of the lemmas above.

Proof of Lemma **??**. The proof goes via structural induction on p.

Basis. Let $p = (V, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, E, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype})$. Assume:

$(p, \sigma) \xrightarrow{a,s,X} (p', \sigma')$

$\Leftrightarrow$ { Rule **??**}

$(v, g, a, (W, r), v') \in E$, $\sigma \models \mathrm{init}(v), \sigma \models g$, $\sigma \models \mathrm{inv}(v), \sigma' \models \mathrm{inv}(v')$,

$\sigma'^{+} \cup \sigma \models r$, $\sigma \restriction_{(X \cup \mathrm{var}_C) \setminus W} = \sigma' \restriction_{(X \cup \mathrm{var}_C) \setminus W}$

$s \equiv a \in \mathrm{act}_S$

$\Leftrightarrow$ {Property of substitution, c does not appear in p, then $a \in p$ iff $a[c/b] \in p[c/b]$}

$(v, g, a[c/b], (W, r), v') \in E[c/b]$, $\sigma \models \mathrm{init}(v), \sigma \models g$, $\sigma \models \mathrm{inv}(v)$,

$\sigma' \models \mathrm{inv}(v')$, $\sigma'^{+} \cup \sigma \models r$, and $\sigma \restriction_{(X \cup \mathrm{var}_C) \setminus W} = \sigma' \restriction_{(X \cup \mathrm{var}_C) \setminus W}$

$s \equiv a[c/b] \in \mathrm{act}_S[c/b]$

$\Leftrightarrow$ {Rule **??**; definition of substitution}

$(p[c/b], \sigma) \xrightarrow{a[c/b],s,X} (p'[c/b], \sigma')$

Induction step. We do a case analysis, depending on the structure of p.

Action scope Assume:

$$(\|[_{\mathrm{A}}\, a :: p\ ]\|, \sigma) \xrightarrow{\ell,s,X} (\|[_{\mathrm{A}}\, a :: p'\ ]\|, \sigma') \tag{B.2}$$

We do a case analysis on the transition that originated (**??**).

Rule ?? was applied to obtain (??) Then we have that $\ell \equiv \tau$, $s \equiv$ false, and we make the following derivation:

$(\|[_{\mathrm{A}}\, a :: p\ ]\|, \sigma) \xrightarrow{\ell,s,X} (\|[_{\mathrm{A}}\, a :: p'\ ]\|, \sigma')$

$\Leftrightarrow$ { Hypothesis, Rule **??**}

$(p, \sigma) \xrightarrow{a,s,X} (p', \sigma')$

$\Leftrightarrow$ { Induction hypothesis}

$(p[c/b], \sigma) \xrightarrow{a[c/b],s,X} (p'[c/b], \sigma')$

$\Leftrightarrow$ { Assumption Rule **??** was applied last; $\tau \neq b$}

$(\|[_{\mathrm{A}}\, a[c/b] :: p[c/b]\ ]\|, \sigma) \xrightarrow{\tau,s,X} (\|[_{\mathrm{A}}\, a[c/b] :: p'[c/b]\ ]\|, \sigma')$

$\Leftrightarrow$ { Definition of substitution }

$(\|[_{\mathrm{A}}\, a :: p\ ]\|\ [c/b], \sigma) \xrightarrow{\tau,s,X} (\|[_{\mathrm{A}}\, a :: p'\ ]\|\ [c/b], \sigma')$

Rule ?? was applied to obtain (??) Then we have that $\ell = a'$, for some $a' \in \mathcal{A}_\tau$ $a' \neq a$, and we make the following derivation:

$(\|[_{\mathrm{A}}\, a :: p\ ]\|, \sigma) \xrightarrow{\ell,s,X} (\|[_{\mathrm{A}}\, a :: p'\ ]\|, \sigma')$

$\Leftrightarrow$ { Hypothesis, Rule **??**}

$(p, \sigma) \xrightarrow{a',s,X} (p', \sigma')$

$\Leftrightarrow$ { Induction hypothesis}

$(p[c/b], \sigma) \xrightarrow{a'[c/b],s,X} (p'[c/b], \sigma')$

$\Leftrightarrow$ { Assumption Rule **??** was applied last; $a' \neq a \wedge c \neq a' \wedge c \neq a \Rightarrow a'[c/b] \neq a[c/b]$ }

$(\llbracket_{\mathrm{A}}\, a[c/b] :: p[c/b] \,\rrbracket, \sigma) \xrightarrow{a'[c/b],s,X} (\llbracket_{\mathrm{A}}\, a[c/b] :: p'[c/b] \,\rrbracket, \sigma')$

$\Leftrightarrow$ { Definition of substitution }

$(\llbracket_{\mathrm{A}}\, a :: p \,\rrbracket\; [c/b], \sigma) \xrightarrow{a'[a/b],s,X} (\llbracket_{\mathrm{A}}\, a :: p' \,\rrbracket\; [c/b], \sigma')$

Parallel composition We make a case analysis depending on whether the action is synchronizing.

Synchronizing action case We make the following derivation:

$(p \parallel q, \sigma) \xrightarrow{a,s,X} (p' \parallel q', \sigma')$

$\Leftrightarrow$ { Assumption Rule **??** was applied }

$(p, \sigma) \xrightarrow{a,\mathrm{true},X} (p', \sigma'), (q, \sigma) \xrightarrow{a,\mathrm{true},X} (q', \sigma')$

$\Leftrightarrow$ { Induction hypothesis }

$(p[c/b], \sigma) \xrightarrow{a[c/b],\mathrm{true},X} (p'[c/b], \sigma')$

$(q[c/b], \sigma) \xrightarrow{a[c/b],\mathrm{true},X} (q'[c/b], \sigma')$

$\Leftrightarrow$ { Assumption Rule **??** was applied; definition of substitution }

$((p \parallel q)[c/b], \sigma) \xrightarrow{a[c/b],s,X} ((p' \parallel q')[c/b], \sigma')$

Interleaving case We make the following derivation:

$(p \parallel q, \sigma) \xrightarrow{a,s,X} (p' \parallel q', \sigma')$

$\Leftrightarrow$ { Assumption Rule **??** was applied last }

$(p, \sigma) \xrightarrow{a,s,X} (p', \sigma')$, $(q, \sigma) \overset{A}{\dashrightarrow} (q', \sigma')$, $a \notin A$

$\Leftrightarrow$ { Induction hypothesis; c does not appear in of $p \parallel q$ therefore $c \notin A$; Lemma **??** }

$(p[[c/b]], \sigma) \xrightarrow{a[c/b],s,X} (p'[c/b], \sigma')$, $(q[c/b], \sigma) \overset{A[c/b]}{\dashrightarrow} (q'[c/b], \sigma')$, $a[c/b] \notin A[c/b]$

$\Leftrightarrow$ { Assumption Rule **??** was applied; definition of substitution }

$((p \parallel q)[c/b], \sigma) \xrightarrow{a[c/b],s,X} ((p' \parallel q')[c/b], \sigma')$

The other interleaving case is symmetric.

Synchronizing action operator We make the following derivation:

$(\gamma_a(p), \sigma) \xrightarrow{a', s \vee a' = a, X} (\gamma_a(p'), \sigma')$

$\Leftrightarrow$ { Rule **??** }

$(p, \sigma) \xrightarrow{a', s, X} (p', \sigma')$

$\Leftrightarrow$ { Induction hypothesis }

$(p[c/b], \sigma) \xrightarrow{a'[c/b], s, X} (p'[c/b], \sigma')$

$\Leftrightarrow$ { Rule **??** }

$(\gamma_{a[c/b]}(p[c/b]), \sigma) \xrightarrow{a'[c/b], s \vee a'[c/b] = a[c/b], X} (\gamma_{a[c/b]}(p'[c/b]), \sigma')$

$\Leftrightarrow$ { Definition substitution, $a' = a \Leftrightarrow a'[c/b] = a[c/b]$}

$(\gamma_a(p)[c/b], \sigma) \xrightarrow{a'[c/b], s \vee a' = a, X} (\gamma_a(p')[c/b], \sigma')$

The remaining cases are straightforward using induction hypothesis.

□

Proof of Lemma **??**. The proof goes via structural induction on the CIF compositions.

Basis. Let $p \equiv (V, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, E, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype})$. Assume:

$(p, \sigma) \overset{A}{\dashrightarrow} (p', \sigma')$ where $A \equiv \mathrm{act}_S$

$\Leftrightarrow$ { Rule **??** }

$\sigma \models \mathrm{init}(v), \sigma \models \mathrm{inv}(v), \sigma' \models \mathrm{inv}(v), \sigma \restriction_{\mathrm{var}_C} = \sigma' \restriction_{\mathrm{var}_C}$

$\Leftrightarrow$ { Rule **??** }

$((V, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, E[c/b], \mathrm{var}_C, \mathrm{act}_S[c/b], \mathrm{dtype}), \sigma)$

$\overset{A}{\dashrightarrow} ((V, \mathrm{id}_v, \mathrm{inv}, \mathrm{tcp}, E[c/b], \mathrm{var}_C, \mathrm{act}_S[c/b], \mathrm{dtype}), \sigma')$

$\Leftrightarrow$ { Definition of substitution }

$(p[c/b], \sigma) \overset{A[c/b]}{\dashrightarrow} (p'[c/b], \sigma')$

Induction step. We do a case analysis depending on the structure of composition p.

Action scope Assume:

$(\llbracket_{\mathrm{A}}\, a :: p \,\rrbracket, \sigma) \overset{A \setminus \{a\}}{\dashrightarrow} (\llbracket_{\mathrm{A}}\, a :: p' \,\rrbracket, \sigma')$

$\Leftrightarrow$ { Rule **??** }

$(p, \sigma) \overset{A}{\dashrightarrow} (p', \sigma')$

$\Leftrightarrow$ { Induction hypothesis }

$(p[c/b], \sigma) \overset{A[c/b]}{\dashrightarrow} (p'[c/b], \sigma')$

$\Leftrightarrow$ { Rule **??** }

$(\llbracket_{\mathrm{A}}\, a[c/b] :: p[c/b] \,\rrbracket, \sigma) \overset{A[c/b] \setminus \{a[c/b]\}}{\dashrightarrow} (\llbracket_{\mathrm{A}}\, a[c/b] :: p'[c/b] \,\rrbracket, \sigma')$

$\Leftrightarrow$ { $A[c/b] \setminus \{a[c/b]\} = (A \setminus \{a\})[c/b]$; definition of substitution }

$(\llbracket_{\mathrm{A}}\, a :: p \,\rrbracket\ [c/b], \sigma) \overset{(A \setminus \{a\})[c/b]}{\dashrightarrow} (\llbracket_{\mathrm{A}}\, a :: p' \,\rrbracket\ [c/b], \sigma')$

Parallel composition Assume:

$(p \parallel q, \sigma) \overset{A_p \cup A_q}{\dashrightarrow} (p' \parallel q', \sigma')$

$\Leftrightarrow$ { Rule **??** }

$(p, \sigma) \overset{A_p}{\dashrightarrow} (p', \sigma'), (q, \sigma) \overset{A_q}{\dashrightarrow} (q', \sigma')$

$\Leftrightarrow$ { Induction hypothesis }

$(p[c/b], \sigma) \overset{A_p[c/b]}{\dashrightarrow} (p'[c/b], \sigma'), (q[c/b], \sigma) \overset{A_q[c/b]}{\dashrightarrow} (q'[c/b], \sigma')$

$\Leftarrow$ { Rule **??**; definition of substitution }

$((p \parallel q)[c/b], \sigma) \overset{(A_p \cup A_q)[c/b]}{\dashrightarrow} ((p' \parallel q')[c/b], \sigma')$

The remaining cases are easy to prove in an analogous way. $\square$

Proof of Property **??**. The proof of the equivalences can be obtained using the obvious bisimulation relations.

For proving the validity of

$$[\![_{\mathrm{A}}\, b :: p\,]\!] \parallel q \leftrightarrow [\![_{\mathrm{A}}\, c :: p[c/b] \parallel q\,]\!]$$

we use as a bisimulation relation:

$$R \triangleq S \cup S^{-1}$$

where relation S is defined as:

$$S \triangleq \{([\![_{\mathrm{A}}\, b :: p\,]\!] \parallel q, [\![_{\mathrm{A}}\, c :: p[c/b] \parallel q\,]\!])\}$$

Assume:

$$([\![_{\mathrm{A}}\, b :: p\,]\!] \parallel q, \sigma) \xrightarrow{a,s,X} (r', \sigma') \tag{B.3}$$

It is easy to see that r' can only be of the form:

$$[\![_{\mathrm{A}}\, b :: p'\,]\!] \parallel q'$$

We do a case analysis depending on whether the action is synchronizing.

Synchronizing case. Then $a \neq b$ because b cannot be synchronizing. Then we have:

$([\![_{\mathrm{A}}\, b :: p\,]\!] \parallel q, \sigma) \xrightarrow{a,s,X} ([\![_{\mathrm{A}}\, b :: p'\,]\!] \parallel q', \sigma')$

$\Leftrightarrow$ { Assumption Rule **??** was applied last }

$([\![_{\mathrm{A}}\, b :: p\,]\!], \sigma) \xrightarrow{a,s,X} ([\![_{\mathrm{A}}\, b :: p'\,]\!], \sigma'), (q, \sigma) \xrightarrow{a,s,X} (q', \sigma')$

$\Leftrightarrow$ { Rule **??** ($a \neq b$) }

$(p, \sigma) \xrightarrow{a,s,X} (p', \sigma'), (q, \sigma) \xrightarrow{a,s,X} (q', \sigma')$

$\Leftrightarrow$ { Induction hypothesis }

$(p[c/b], \sigma) \xrightarrow{a[c/b],s,X} (p'[c/b], \sigma'), (q, \sigma) \xrightarrow{a,s,X} (q', \sigma')$

$\Leftrightarrow$ { $a \neq b \Rightarrow a[c/b] = a$; Rule **??** }

$(p[c/b] \parallel q, \sigma) \xrightarrow{a,s,X} (p'[c/b] \parallel q', \sigma')$

$\Leftrightarrow$ { Rule **??**; c does not occur in p therefore $c \neq a$ }

$([\![_{\mathrm{A}}\, c :: p[c/b] \parallel q\,]\!], \sigma) \xrightarrow{a,s,X} ([\![_{\mathrm{A}}\, c :: p'[c/b] \parallel q'\,]\!], \sigma')$

Asynchronous case. We distinguish three sub-cases, depending on the component that performed the action, and whether this action was hidden by the variable scope operator at the top level.

The first component performed a non-hidden action Then $b \neq a$, since otherwise action a would have been hidden.

Then we make the following derivation:

$(|[_{\mathrm{A}}\, b :: p\,]| \| q, \sigma) \xrightarrow{a,s,X} (|[_{\mathrm{A}}\, b :: p'\,]| \| q', \sigma')$

$\Leftrightarrow$ { Assumption Rule **??** was applied last }

$(|[_{\mathrm{A}}\, b :: p\,]|, \sigma) \xrightarrow{a,s,X} (|[_{\mathrm{A}}\, b :: p'\,]|, \sigma')$, $(q, \sigma) \overset{A}{\dashrightarrow} (q', \sigma')$, $a \notin A$

$\Leftrightarrow$ { Rule **??**; $a \neq b$ }

$(p, \sigma) \xrightarrow{a,s,X} (p', \sigma')$, $(q, \sigma) \overset{A}{\dashrightarrow} (q', \sigma')$, $a \notin A$

$\Leftrightarrow$ { Lemma **??**; $a \neq b$; definition of substitution }

$(p[c/b], \sigma) \xrightarrow{a,s,X} (p'[c/b], \sigma')$, $(q, \sigma) \overset{A}{\dashrightarrow} (q', \sigma')$, $a \notin A$

$\Leftrightarrow$ { Assumption Rule **??** was applied last }

$(p[c/b] \parallel q, \sigma) \xrightarrow{a,s,X} (p'[c/b] \parallel q', \sigma')$

$\Leftrightarrow$ { Rule **??**; $a \neq b$, $c \neq a$ }

$(|[_{\mathrm{A}}\, a :: p[c/b] \parallel q\,]|, \sigma) \xrightarrow{a,s,X} (|[_{\mathrm{A}}\, c :: p'[c/b] \parallel q'\,]|, \sigma')$

The first component performed a hidden action In this case we have $a = \tau$, and b is the action generated by p.

Then we make the following derivation:

$(|[_{\mathrm{A}}\, b :: p\,]| \| q, \sigma) \xrightarrow{a,s,X} (|[_{\mathrm{A}}\, b :: p'\,]| \| q', \sigma')$

$\Leftrightarrow$ { Assumption Rule **??** was applied last }

$(|[_{\mathrm{A}}\, b :: p\,]|, \sigma) \xrightarrow{\tau,s,X} (|[_{\mathrm{A}}\, b :: p'\,]|, \sigma')$, $(q, \sigma) \overset{A}{\dashrightarrow} (q', \sigma')$, $a \notin A$

$\Leftrightarrow$ { Rule **??** }

$(p, \sigma) \xrightarrow{b,s,X} (p', \sigma')$, $(q, \sigma) \overset{A}{\dashrightarrow} (q', \sigma')$, $a \notin A$

$\Leftrightarrow$ { Lemma **??** }

$(p[c/b], \sigma) \xrightarrow{b[c/b],s,X} (p'[c/b], \sigma')$, $(q, \sigma) \overset{A}{\dashrightarrow} (q', \sigma')$, $a \notin A$

$\Leftrightarrow$ { Rule **??**; $b[c/b] = c$}

$(p[c/b] \parallel q, \sigma) \xrightarrow{c,s,X} (p'[c/b] \parallel q, \sigma')$

$\Leftrightarrow$ { Rule **??**; $a = \tau$}

$([\![_{\mathrm{A}}\, a :: p[c/b] \parallel q\,]\!], \sigma) \xrightarrow{a,s,X} ([\![_{\mathrm{A}}\, c :: p'[c/b] \parallel q'\,]\!], \sigma')$

The second component performed an action We make the following derivation:

$([\![_{\mathrm{A}}\, b :: p\,]\!] \parallel q, \sigma) \xrightarrow{a,s,X} ([\![_{\mathrm{A}}\, b :: p'\,]\!] \parallel q', \sigma')$

$\Leftrightarrow$ { Assumption Rule **??** was applied last }

$([\![_{\mathrm{A}}\, b :: p\,]\!], \sigma) \overset{A\setminus\{b\}}{\dashrightarrow} ([\![_{\mathrm{A}}\, b :: p'\,]\!], \sigma')$, $(q, \sigma) \overset{a,s,X}{\dashrightarrow} (q', \sigma')$, $a \notin A \setminus \{b\}$

$\Leftrightarrow$ { Rule **??** }

$(p, \sigma) \overset{A}{\dashrightarrow} (p', \sigma')$, $(q, \sigma) \xrightarrow{a,s,X} (q', \sigma')$, $a \notin A \setminus \{b\}$

$\Leftrightarrow$ { Lemma **??**; $c \neq a \Rightarrow (a \notin A \setminus \{b\} \Leftrightarrow a \notin A[c/b])$}

$(p[c/b], \sigma) \overset{A[c/b]}{\dashrightarrow} (p'[c/b], \sigma')$, $(q, \sigma) \xrightarrow{a,s,X} (q', \sigma')$, $a \notin A[c/b]$

$\Leftrightarrow$ { Assumption Rule **??** was applied last }

$(p[c/b] \parallel q, \sigma) \xrightarrow{a,s,X} (p'[c/b] \parallel q', \sigma')$

$\Leftrightarrow$ { Rule **??**; $c \neq a$ }

$([\![_{\mathrm{A}}\, c :: p[c/b] \parallel q\,]\!], \sigma) \xrightarrow{a,s,X} ([\![_{\mathrm{A}}\, c :: p'[c/b] \parallel q'\,]\!], \sigma')$

For the reminder bisimilarities the proof that the transfer conditions hold can be obtained in a similar manner. $\square$